UNIVERSITAT DE BARCELONA

Treball final de grau

GRAU DE MATEMÀTIQUES I INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Estudi dels mètodes d'optimització al Deep Reinforcement Learning aplicat als videojocs

Autor: Xavier de Juan Pulido

Director: Dr. Eloi Puertas Prats, Eloi Sans Gispert
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, 20 de juny de 2021

Abstract

This report studies the Reinforcement Learning problem based on the *Sonic the Hedgehog™* video game franchise. Fundamental concepts of RL are introduced in order to formalize the problem and present some state of the art Deep Reinforcement Learning (DRL) algorithms to solve it. It is also studied the core optimization problem from deep learning. In addition, some mathematical background is presented behind the optimization problem and behind the solutions based on Gradient Descent optimization algorithm. The aim is to empirically show that Adam is the best choice of optimizer for DRL methods in the Sonic benchmark. In the course of achieving it, first it has been explained the main elements of reinforcement learning and how this kind of problems can be formulated as Markov Decision Processes. Then it has been examined its combination with deep learning, also called Deep Reinforcement Learning, and different algorithms are provided to exemplify how they are combined. The second part of this work studies the mathematical background behind optimization in deep learning. This work also considers the optimization problem that appears when wanting to update the parameters of a neural network and a basic solution, the Gradient Descent optimization algorithm. Different variants of Gradient Descent have been looked at, along with the analysis of its challenges and the most common optimization algorithms that improve the basic Gradient Descent. During the experimentation phase, a set of Sonic levels has been used, splitted in train and test levels, to define each environment and common metrics to evaluate the experiments score and time. Each experiment consist in a DRL algorithm, between Joint PPO and Rainbow, and an optimization method between Adam, Adadelta, AdaGrad, Nesterov Accelerated Gradient and Momentum. Results show that Adam is the best choice of optimizer for both DRL algorithms, making the agents achieve better scores and learn to generalize between different levels. That empirically proves that Adam is the best optimizer for Joint PPO and Rainbow on the Sonic benchmark.

Resum

Aquest treball estudia el problema de l'aprenentatge per reforç basat en la franquícia de videojocs del *Sonic the Hedgehog™*. S'introdueixen els conceptes fonamentals de l'aprenentatge per reforç per formalitzar el problema i es presenten alguns algorismes d'aprenentatge per reforç profund (DRL en anglès) d'última generació per a resoldre'l. També s'estudia el problema central d'optimització de l'aprenentatge profund. Així mateix, es presenta el rerefons matemàtic d'aquest problema d'optimització i de les solucions basades en l'algorisme d'optimització Gradient Descent. L'objectiu és demostrar empíricament que l'Adam és la millor elecció d'optimitzador per als mètodes DRL a l'entorn de referència del Sonic. Per aconseguir-ho, primer s'expliquen els principals elements de l'aprenentatge per reforç i com, un problema d'aquest tipus, pot formular-se com un Procés de Decisió de Markov. Després s'examina la seva combinació amb l'aprenentatge profund, també anomenat aprenentatge per reforç profund, i es proporcionen diferents algorismes per exemplificar aquesta combinació. A la segona part d'aquest treball s'estudia el rerefons matemàtic de l'optimització en l'aprenentatge profund. S'introdueix el problema d'optimització que apareix al voler actualitzar els paràmetres d'una xarxa neuronal i una solució bàsica, l'algorisme d'optimització Gradient Descent. El treball presenta diferents variants del Gradient Descent, juntament amb l'anàlisi dels seus reptes i els algorismes d'optimització més comuns que milloren el Gradient Descent bàsic. Durant la fase d'ex-

perimentació s'utilitza un conjunt de nivells del Sonic, dividits en nivells d'entrenament i d'avaluació, per definir cada entorn i unes mètriques comunes per avaluar la puntuació i temps dels experiments. Cada experiment consisteix en un algorisme DRL, entre el Joint PPO i el Rainbow, i un mètode d'optimització entre l'Adam, l'Adadelta, l'AdaGrad, el Nesterov Accelerated Gradient i el Momentum. Els resultats mostren que l'Adam és la millor elecció d'optimitzador per a tots dos algorismes DRL, fent que els agents aconsegueixin millors puntuacions i aprenguin a generalitzar entre diferents nivells. Això demostra empíricament que l'Adam és el millor optimitzador pel Joint PPO i pel Rainbow a l'entorn de referència del Sonic.

Resumen

Este trabajo estudia el problema del aprendizaje por refuerzo basado en la franquicia de videojuegos del *Sonic the Hedgehog™*. Se introducen los conceptos fundamentales del aprendizaje por refuerzo para formalizar el problema y se presentan algunos algoritmos de aprendizaje por refuerzo profundo (DRL en inglés) de última generación para resolverlo. También se estudia el problema central de optimización del aprendizaje profundo. Y, además, se presenta el trasfondo matemático detrás de este problema de optimización y detrás de las soluciones basadas en el algoritmo de optimización Gradient Descent. El objetivo es demostrar empíricamente que Adam es la mejor elección de optimizador para los métodos DRL en el entorno de referencia del Sonic. Para conseguirlo, primero se explican los principales elementos del aprendizaje por refuerzo y cómo un problema de este tipo puede formularse como un Proceso de Decisión de Markov. Luego se examina su combinación con el aprendizaje profundo, también llamado aprendizaje por refuerzo profundo, y se proporcionan diferentes algoritmos para ejemplificar esta combinación. En la segunda parte de este trabajo se estudia el trasfondo matemático de la optimización en el aprendizaje profundo. Se introduce el problema de optimización que aparece al querer actualizar los parámetros de una red neuronal y una solución básica, el algoritmo de optimización Gradient Descent. El trabajo presenta distintas variantes del Gradient Descent, junto con el análisis de sus retos y los algoritmos de optimización más comunes que mejoran el Gradient Descent básico. Durante la fase de experimentación se utiliza un conjunto de niveles del Sonic, divididos en niveles de entrenamiento y de evaluación, para definir cada entorno y unas métricas comunes para evaluar la puntuación y tiempo de los experimentos. Cada experimento consiste en un algoritmo DRL, entre Joint PPO y Rainbow, y un método de optimización entre Adam, Adadelta, AdaGrad, Nesterov Accelerated Gradient y Momentum. Los resultados muestran que Adam es la mejor elección de optimizador para ambos algoritmos DRL, haciendo que los agentes consigan mejores puntuaciones y aprendan a generalizar entre diferentes niveles. Esto demuestra empíricamente que Adam es el mejor optimizador para Joint PPO y Rainbow en el entorno de referencia del Sonic.

2020 Mathematics Subject Classification. 90C06, 90C25, 90C26, 90C39, 90C40, 90C52

Agraïments

D'una banda, vull agrair al Dr Eloi Puertas i a l'Eloi Sans haver acceptat dirigir aquest treball i el suport donat durant el transcurs d'aquest treball. Agraeixo molt totes les directrius, comentaris i recomanacions que m'han permès realitzar aquest treball. D'altra banda, vull donar les gràcies a la meva família, parella i amics per haver cregut en mi durant tota la carrera i haver-me animat en els moments difícils. Per últim, m'agradaria destacar l'agraïment que sento cap als meus pares els quals m'han permès el privilegi d'estudiar la carrera que he volgut subvencionant els costos d'aquesta.

Índex

Capítol 1

Introducció

Durant els últims anys, el *Reinforcement Learning (RL)*[1] (aprenentatge per reforç) ha guanyat popularitat gràcies a la incorporació del *Deep Learning*[2] (aprenentatge profund), donant lloc a una nova disciplina, el *Deep Reinforcement Learning (DRL)*[3] (aprenentatge per reforç profund). Aquesta ha permès resoldre una àmplia gamma de tasques complexes de presa de decisions que, anteriorment, no es trobaven a l'abast d'una màquina.

El reinforcement learning considera el problema d'un agent que aprèn a prendre decisions en un entorn a partir de la prova i error. El deep reinforcement learning incorpora les xarxes neuronals a la solució permetent treballar amb problemes de grans dimensions. Generalment, els agents de DRL reben entrades de grans dimensions a cada pas, per exemple els píxels de la pantalla d'un videojoc, i realitzen accions en funció d'una política basada en xarxes neuronals profundes. Aquest mecanisme d'aprenentatge actualitza la política amb l'objectiu de maximitzar les recompenses retornades per l'entorn.

En aquest treball ens centrarem en les aplicacions del DRL als jocs, especialment en els videojocs, una àrea de recerca amb una llarga trajectòria. La recerca en els videojocs estudia com utilitzar els algorismes DRL amb l'objectiu d'aconseguir rendiments iguals o superiors als dels éssers humans en els videojocs. La característica principal dels videojocs és la construcció d'un entorn virtual segur i controlat que, amb la interacció, permet generar un nombre infinit de dades a una velocitat superior que en un entorn real. Aquesta característica converteix els videojocs en bancs de proves excel·lents per a la recerca en aquesta àrea però, també, per altres tipus de tasques. Per exemple, l'aprenentatge en un entorn simulat en lloc d'un entorn real permet accelerar la velocitat d'aprenentatge i reduir possibles danys materials. També permet generar conjunts de dades per altres tasques com per exemple imatges per la visió per computador. O, inclús, la millora del desenvolupament dels videojocs per la detecció de *bugs* o generació de personatges no controlats per jugadors més intel·ligents amb comportaments més naturals.

Així doncs, aquest tipus d'aprenentatge ha destacat, principalment, per la seva aplicació en els jocs/videojocs[4] i robòtica[5]. Tot i això, el DRL té una àmplia gamma d'aplicacions en les quals s'utilitzen solucions basades en el reinforcement learning com per exemple el *Natural Language Processing*[6] (processament de llenguatge natural), el *Computer Vision*[7] (visió per computador), les finances[8] i la gestió empresarial[9].

Pel que fa a l'aprenentatge en videojocs, l'aparició del DRL va millorar dràsticament la capacitat de generalització i escalabilitat dels algorismes tradicionals de RL. El DRL ha aconseguit grans progressos en jocs com per exemple l'*AlphaZero*[10] el qual va aconseguir,

en només 24 hores, un rendiment superhumà en els escacs, el shogi i el Go vencent els campions mundials de cada un d'ells, i el *Libratus*[11] vencent campions del Texas Hold'em poker. En el sector dels videojocs ha mostrat grans èxits incloent 2600 jocs de l'*Atari*[12], la plataforma *ViZDoom*[13] basada en el joc Doom en la qual es van aconseguir agents comparables al nivell humà, l'*AlphaStar*[1] el qual va guanyar jugadors professionals del videojoc *StarcCraft 2* i, per últim, l'*OpenAI Five*[2] per jugar al videojoc *Dota 2* en format 5 contra 5 aconseguint vèncer els campions mundials d'aquest videojoc.

Aquest treball es centra en l'aplicació del DRL en un videojoc en particular, la versió clàssica del *Sonic the Hedgehog*™ [3] per a la videoconsola SEGA™. L'any 2018 l'empresa OpenAI va presentar un concurs, el *Retro Contest*[14], el qual tenia com a objectiu explorar el desenvolupament d'algorismes DRL que poguessin generalitzar a partir de l'experiència anterior. El concurs es basava en el videojoc del Sonic i es va proporcionar un codi base de referència el qual ha sigut utilitzat com a base d'aquest treball. D'aquesta manera, el Retro Contest s'utilitzarà com a referència per desenvolupar aquest treball.

Aquest treball presentarà dos problemes dins del context del DRL. D'una banda, l'aplicació al videojoc del Sonic seguint la línia d'investigació del Retro Contest. D'altra banda, el problema d'optimització inherent al deep learning. Com hem vist, el DRL incorpora les xarxes neuronals a la solució. El mecanisme d'aprenentatge de la solució consisteix, en gran mesura, en ajustar els paràmetres de la xarxa neuronal basant-se en l'optimització d'una funció de cost, donant lloc al problema d'optimització. A efectes pràctics, serà interessant veure com es desenvolupen els mètodes d'optimització dins l'aplicació del DRL al videojoc del Sonic.

1.1 Objectius

Passem doncs a introduir l'objectiu principal del treball així com objectius més específics que voldrem assolir en el desenvolupament d'aquest treball.

L'objectiu principal d'aquest treball consistirà en analitzar l'efecte dels mètodes d'optimització en el rendiment de diferents algorismes DRL en el videojoc del Sonic. Més detalladament, d'una banda estudiarem el problema d'optimització i l'evolució dels mètodes durant els últims anys i, d'altra banda, analitzarem el problema de reinforcement learning en base al videojoc del Sonic mostrant dos algorismes d'última generació en el camp del DRL, el *PPO*[15] i el *Rainbow*[16].

A efectes pràctics, per cada algorisme DRL, avaluarem el rendiment d'aquest utilitzant diferents algorismes d'optimització. En particular utilitzarem els següents algorismes d'optimització: *Momentum*[17], *Nesterov Accelerated Gradient*[18], *AdaGrad*[19], *Adadelta*[20] i Adam[21]. Com veurem més endavant, aquests algorismes representen una evolució de l'algorisme d'optimització base, el *Gradient Descent*[22]. D'entre aquests optimitzadors, podem destacar que l'Adam es troba al cim de l'evolució i, per aquest motiu, podem esperar que serà el que assolirà millors resultats en l'avaluació dels experiments.

A més, per donar suport a l'objectiu principal del treball definirem els següents objectius específics:

- Mostrar les bases del reinforcement learning i la combinació amb el deep learning,

[1] https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii
[2] https://openai.com/projects/five/
[3] https://www.sega.com/games/sonic-hedgehog

el deep reinforcement learning.

- Exposar el problema d'optimització en el deep learning així com alguns dels mètodes més utilitzats.
- Analitzar el funcionament i convergència de l'algorisme d'optimització més popular, l'Adam.
- Introduir una de les plataformes de recerca del reinforcement learning en videojocs, el *Gym Retro*[14].
- Comprovar el rendiment d'algorismes DRL d'última generació, com el Joint PPO i el Rainbow, en el Sonic.
- Verificar, empíricament, que els algorismes DRL donen millors resultats quan s'utilitza l'algorisme d'optmització Adam.

1.2 Estructura del treball

La distribució de la resta del treball s'organitzarà de la següent manera. A la secció 2 definirem el reinforcement learning, el problema que planteja i com resoldre'l. A la mateixa secció també definirem el deep reinforcement learning fent una petita introducció al deep learning. A la secció 3 estudiarem el problema a tractar així com el presentat en el Retro Contest veient-ne els elements que conformen el problema. I, a continuació, farem un anàlisi de la manera en què volem assolir l'objectiu principal del treball. La secció 4 es centrarà en l'estudi teòric dels mètodes d'optimització. Presentarem el problema d'optimització, l'algorisme bàsic de resolució i l'evolució d'aquest algorisme centrant-nos, en última instància, en l'algorisme Adam. Tot seguit, la secció 5 presentarà la configuració del problema enfocat a la pràctica. Definirem els algorismes DRL i d'optimització, les dades, entorn i mètriques que usarem per realitzar els experiments. La secció 6 mostrarà els resultats derivats dels experiments. En aquesta secció farem un anàlisi i comparació d'aquests resultats. Per acabar, a la secció 7 mostrarem les conclusions del treball així com possibles treballs futurs.

Capítol 2

Context

El Reinforcement Learning (aprenentatge per reforç) és una àrea del *Machine Learning*[23] (aprenentatge automàtic) basada en la idea d'aprendre a partir de la interacció. Intuïtivament, consisteix en realitzar un procés de prova i error afegint l'aprenentatge del resultat d'aquestes accions.

Més detalladament, el reinforcement learning consisteix en un mapatge de situacions i accions per maximitzar una recompensa. Definirem com a agent, l'ésser que es desenvolupa a l'entorn, és a dir l'ésser que realitza les accions en un entorn donat. A aquest no cal explicar-li quines accions ha de prendre sinó que ha de descobrir quines accions generen més recompenses en provar-les. En determinades situacions, la recompensa d'una acció no només pot afectar la recompensa immediata sinó també la següent i, en conseqüència totes les posteriors. Aquestes dues característiques, cerca de prova i error i recompensa retardada, són les característiques distintives i més importants del reinforcement learning. Aquest tipus d'algorismes també requereixen d'una combinació sinèrgica entre cerca i memòria per tal d'explorar bones accions i, a més, recordar quines accions han funcionat bé anteriorment.

Un dels reptes que sorgeixen en aquest tipus de problemes és considerar el balanç entre exploració i explotació. Per aconseguir la màxima recompensa, l'agent preferirà realitzar accions que ha provat anteriorment i sap que són bones en termes de recompensa. Tot i això, per descobrir aquestes accions, l'agent ha de provar accions que no ha seleccionat anteriorment. Per tant, l'agent ha d'explotar les accions que coneix i sap que donen bons resultats a la vegada que explora noves situacions per realitzar accions millors en un futur.

2.1 Elements del Reinforcement Learning

Havent introduït el problema, anem a veure els elements principals del reinforcement learning. Més enllà de l'agent i l'entorn, podem identificar quatre subelements principals d'un sistema de reinforcement learning: una *política*, un *senyal de recompensa*, una *funció de valor* i, opcionalment, un *model* de l'entorn.

La política defineix la manera de comportar-se de l'agent en un moment determinat. Podríem dir que consisteix en un mapatge des dels estats percebuts a les accions a realitzar quan es troben en aquests estats. Podem afirmar que es tracta del nucli de l'agent en el sentit que és suficient per determinar el seu comportament.

El senyal de recompensa defineix l'objectiu del problema, indicant què és bo en un sentit immediat. A cada pas de temps, l'entorn envia a l'agent un número únic, una recompensa. L'únic objectiu de l'agent és maximitzar la recompensa total que rep a la llarga. La recompensa enviada a l'agent depèn de l'estat actual de l'agent i l'acció realitzada en un determinat moment. L'agent no pot alterar aquest procés però pot influir-hi amb les seves accions, arribant a tenir un efecte directe sobre la recompensa o un efecte indirecte canviant l'estat de l'entorn. El senyal de recompensa és la base principal per modificar la política.

La funció de valor, a diferència del senyal de recompensa, especifica què és bo a llarg termini. És a dir, el valor d'un estat és la quantitat total de recompenses que un agent pot esperar acumular en el futur partint de l'estat actual o una predicció d'aquest total.

Per últim, el model de l'entorn. El model imita el comportament de l'entorn o, més generalment, permet fer inferències sobre com es comportarà l'entorn. Aquests models s'utilitzen per a la planificació, així, el fet de decidir sobre una línia d'actuació té en compte les possibles situacions futures abans de ser experimentades. El model de l'entorn ens permet diferenciar dos tipus de mètodes per resoldre problemes de reinforcement learning, els basats en models (model-based) i els lliures de models (model-free).

2.2 Procés de Decisió de Markov

Un *Markov Decision Process*[24] (Procés de Decisió de Markov) o MDP ens permet formalitzar el problema de reinforcement learning descrit anteriorment. Per tant, ens referirem a aquest tipus de problemes com MDPs i veurem com es defineixen.

Els MDP consisteixen essencialment en estats, accions, transicions entre estats i una funció de recompensa. Generalment, els MDP ens permeten modelar problemes que podrien tenir infinits estats i/o accions, tot i això, ens limitarem als problemes amb estats i accions finites. Abans de definir formalment en què consisteix un MDP cal definir el context on es situa el problema i els seus elements.

Agent i Entorn Tal i com hem vist a la secció anterior, el problema a modelar consisteix essencialment en aprendre des de la interacció per assolir un objectiu. Així doncs, ens referirem a l'ésser que aprèn i pren les decisions com l'*agent*. Tot amb el que l'agent interactua i no pertany a ell és al que anomenarem *entorn*. Aquests dos elements interactuen entre ells contínuament, l'agent seleccionant i realitzant accions i l'entorn responent a aquestes accions, presentant noves situacions a l'agent i donant recompenses a l'agent.

Estats El conjunt d'estats de l'entorn S es defineix com un conjunt finit de dimensió N, Un *estat* és una caracterització única de tot allò que és important en un estat del problema modelat.

Accions El conjunt d'accions A es defineix com un conjunt finit de dimensió K. Les accions s'utilitzen per controlar l'estat del sistema. Generalment, el conjunt d'accions que es poden aplicar en un estat particular és el conjunt A, però en determinats problemes, es pot donar la situació que el conjunt d'accions disponibles sigui un subconjunt d'aquest conjunt A.

La Funció de Transició En aplicar una acció $a \in A$ en un estat $s \in S$, el sistema fa una *transició*, de $s \in S$ a un nou estat $s' \in S$, basada en una distribució de probabilitats

sobre el conjunt de transicions possibles. D'aquesta manera, definim la *funció de transició* com $T : S \times A \times S \to [0, 1]$, és a dir, la probabilitat d'acabar en un estat s' havent realitzat l'acció a des de l'estat s es denota per $T(s, a, s')$.

Diem que el sistema que s'està controlant és *Markovià* o que té la *propietat de Markov* si el resultat d'una acció no depèn de les accions ni estats anteriors, és a dir, només depèn de l'estat actual. La idea de la dinàmica Markoviana és que l'estat actual proporciona suficient informació per realitzar una decisió òptima sense tenir en compte els estats que el precedien.

La funció de recompensa Aquesta funció especifica les recompenses per ser en un estat o per realitzar una acció en un estat. Es defineix com $R : S \to \mathbb{R}$ especificant, amb un valor numèric, les recompenses d'un estat. D'altra banda, aquesta funció es pot definir de dues maneres diferents: $R : S \times A \to \mathbb{R}$ o $R : S \times A \times S \to \mathbb{R}$. La primera proporciona recompenses en realitzar una acció des d'un estat i la segona proporciona recompenses per transicions particulars entre estats. Totes tres definicions són intercanviables, tot i això, la tercera sol ser més convenient en els algorismes lliures de models, ja que normalment es requereixen tant l'estat inicial com l'estat resultant per guardar còpies dels valors.

La funció de recompensa és un element important d'un MDP ja que, implícitament, defineix l'objectiu de l'aprenentatge. L'agent és guiat per les recompenses que rep amb l'objectiu de maximitzar-les, per aquest motiu, la funció de recompensa s'utilitza per guiar la direcció per la qual el sistema haurà de ser controlat.

Procés de decisió de Markov Un procés de decisió de Markov (finit) és una tupla $\langle S, A, T, R\rangle$ on S és un conjunt finit d'estats, A és un conjunt finit d'accions, T és la funció de transició definida per $T : S \times A \times S \to [0, 1]$ i R és la funció de recompensa definida per $R : S \times A \times S \to \mathbb{R}$. En particular, un MDP té la propietat de Markov.

Objectiu de l'agent Fins ara, hem fet referència a l'objectiu de l'agent, d'una manera informal, com maximitzar la recompensa acumulada. Vegem doncs, formalment, què significa això. Denotem la seqüència de recompenses a partir del pas de temps t com $R_{t+1}, R_{t+2}, R_{t+3}, \ldots$. En general busquem maximitzar el *retorn esperat*, G_t, definit per una funció específica de la seqüència de recompenses. El cas més simple consisteix en la suma de recompenses: $G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T$ on T és un pas de temps final. Aquesta aproximació té sentit quan existeix un estat final, és a dir, quan podem trencar la interacció agent-entorn en subseqüències a les quals anomenem *episodis*. Cada episodi acaba en un estat especial anomenat *estat terminal* seguidament d'un restabliment de l'entorn a un estat inicial. Aquest tipus de tasques les anomenem *episodic tasks*, en les quals cal diferenciar el conjunt d'estats no terminals del conjunt de tots els estats juntament amb l'estat terminal.

D'altra banda, els casos en què l'entorn no conté cap estat terminal, és a dir no es pot trencar la interacció agent-entorn en episodis, els anomenarem *continuing tasks*. En aquests casos, la definició G_t anterior és problemàtica ja que el pas de temps final seria $T = \infty$ i G_t, el qual volem maximitzar, també podria ser infinit. Per exemple, si l'agent rep una recompensa $+1$ a cada pas de temps $G_t \to \infty$.

Així doncs, el concepte que necessitem és el factor de *descompte*. D'aquesta manera, el retorn que intentarà maximitzar l'agent serà el *retorn amb descompte*: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ on $0 \leq \gamma \leq 1$ anomenat *taxa de descompte*.

Això ens permet que, si $\gamma < 1$ la suma té un valor finit i ens permet, ajustant el valor

de γ, donar més importància a les recompenses immediates que no pas a les futures.

Polítiques Donat un MDP $\langle S, A, T, R\rangle$, una *política* és una funció que retorna, per cada estat $s \in S$ una acció $a \in A$ (o pertanyent a un subconjunt d'A). Formalment, una *política determinista* π és una funció definida com $\pi : S \to A$. També és possible definir una *política estocàstica* com $\pi : S \times A \to [0, 1]$ tal que per cada estat $s \in S$, $\pi(s, a) \geq 0$ i $\sum_{a\in A} \pi(s, a) = 1$. D'aquesta manera, en el problema donat, ens situem en un estat inicial, la política π suggereix una acció, basant-nos en T i R es realitza la transició i l'agent rep una recompensa. Aquest procés es repeteix fins arribar a un estat terminal o es repeteix indefinidament en el cas en que no existeixi estat terminal.

Funcions de valor Fins ara, hem definit un MDP i hem vist com l'agent interactua amb l'entorn mitjançant una política per aconseguir el seu objectiu, maximitzar una recompensa. Per assolir aquest objectiu, és important escollir una política òptima. La majoria d'algorismes per MDPs calculen aquesta política òptima utilitzant les funcions de valor. Les funcions de valors representen *quant bo* és, per l'agent, trobar-se en un determinat estat o realitzar una acció en un determinat estat. Aquesta noció de *quant bo* s'expressa en termes del retorn esperat.

El valor d'un estat s sota una política π, denotat per $V^\pi(s)$ és el retorn esperat quan comencem a s i seguim π. En aquesta secció utilitzarem el retorn amb descompte, d'aquesta manera $V^\pi : S \to \mathbb{R}$ definida per:

$$V^\pi(s) = E_\pi\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s \right\} \tag{2.2.1}$$

Una *funció de valor estat-acció* similar $Q : S \times A \to \mathbb{R}$ es pot definir com el retorn esperat començant a l'estat s, realitzant l'acció a i seguint la política π:

$$Q^\pi(s, a) = E_\pi\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a \right\} \tag{2.2.2}$$

Una propietat fonamental de les funcions de valor és que satisfan certes propietats de recursivitat. D'aquesta manera podem expressar l'equació 2.2.1 en termes de l'*Equació de Bellman*[25]:

$$V^\pi(s) = \sum_{s'\in S} T(s, \pi(s), s')\Big(R(s, \pi(s), s') + \gamma V^\pi(s') \Big) \tag{2.2.3}$$

L'objectiu en qualsevol MDP donat és trobar la millor política, és a dir, la que proporcioni majors recompenses. Això significa que maximitzar la funció de valor de l'equació 2.2.1 per tots els estats $s \in S$. La *política òptima*, denotada per π^*, és aquella que per tot estat $s \in S$ i tota política π, $V^{\pi^*} \geq V^\pi$. Per tant, la solució òptima $V^* = V^{\pi^*}$ satisfà

$$V^*(s) = \max_{a\in A} \sum_{s'\in S} T(s, a, s')\Big(R(s, a, s') + \gamma V^*(s') \Big) \tag{2.2.4}$$

A la qual anomenem *Equació d'optimalitat de Bellman.* D'aquesta manera, podem escollir com a política òptima aquella que selecciona l'acció òptima donada la funció de valor V^* com:

$$\pi^*(s) = \arg\max_{a\in A} \sum_{s'\in S} T(s, a, s')\Big(R(s, a, s') + \gamma V^*(s') \Big) \tag{2.2.5}$$

A aquesta la anomenem la *política greedy* (política voraç), denotada per $\pi_{greedy}(V)$ ja que selecciona la millor acció usant la funció de valor V. Anàlogament, podem definir el valor òptim estat-acció com:

$$Q^*(s,a) = \sum_{s' \in S} T(s,a,s') \Big(R(s,a,s') + \gamma \max_{a' \in A} Q^*(s',a') \Big) \tag{2.2.6}$$

De la mateixa manera, definim la política òptima com:

$$\pi^*(s) = \arg\max_{a \in A} Q^*(s,a) \tag{2.2.7}$$

I, de nou, podríem definir una política greedy basada en Q, denotada per $\pi_{greedy}(Q)$ que, en contrast amb $\pi_{greedy}(V)$, veiem que no hi ha necessitat de consultar el model del MDP sinó que la Q-funció és suficient.

Finalment, podem veure que les Q-funcions són útils ja que fan que la suma ponderada de diferents alternatives mitjançant la funció de transició sigui innecessària. Aquesta és la raó per la qual les aproximacions lliures de models, és a dir en el cas que les funcions T i R siguin desconegudes, les Q-funcions són apreses en lloc de les V-funcions. Per últim, podem veure que, la relació entre Q^* i V^* ve donada per la igualtat:

$$V^*(s) = \max_a Q^*(s,a) \tag{2.2.8}$$

2.3 Resoldre MDPs

Ara que ja hem definit el nostre problema, modelant-lo com un MDP, hem vist què són les polítiques i les funcions de valor, és el moment de veure com trobar la política òptima del MDP donat. Solucionar un MDP, en essència, consisteix en calcular la política òptima π^*. Existeixen diverses distincions pels que fa als algorismes que s'han desenvolupat fins ara, tot i això, la distinció més important és la que diferencia algorismes basats en models i algorismes lliures de models.

Un mecanisme subjacent important és l'anomenat *generalized policy iteration o GPI* (iteració de política generalitzada) present en tots els mètodes. Consisteix en dos processos d'interacció: l'avaluació de la política i la millora de la política. Tal i com podem intuir, el primer pas consisteix en avaluar la política, és a dir, avaluar V^π que ens permet recollir informació pel següent pas, la millora de la política. En aquest segon estat s'avaluen els valors de les accions per cada estat amb l'objectiu de trobar possibles millores, és a dir, possibles accions en un estat particular que siguin millors que les de l'actual política.

2.3.1 Dynamic Programming: Solucions basades en models

Els algorismes basats en models existeixen sota el nom general *Dynamic Programming o DP*[25]. La suposició bàsica en aquesta classe d'algorismes és el coneixement previ d'un model del MDP i es pot utilitzar per calcular funcions de valor i polítiques mitjançant l'equació de Bellman. Els mètodes d'aquesta secció assumeixen un MDP estàndard $\langle S, A, T, R \rangle$ amb conjunts d'estats i accions finits i discrets tal que poden ser guardats en taules. A més, s'assumeix que les funcions de transició, recompensa i valor guarden els valors per tots els estats i accions per separat.

Els dos mètodes bàsics de DP són *policy iteration*[26] (iteració de polítiques) i *value iteration*[25] (iteració de valors). En el primer, el mecanisme GPI està clarament separat en dos passos mentre que en el segon hi ha una estreta integració entre l'avaluació i la millora de la política.

Policy Iteration Itera entre les dues fases del GPI. Durant la fase d'avaluació de la política es calcula la funció de valor per la política actual i en la fase de millora de la política es calcula una política millorada mitjançant la maximització de la funció de valor. Aquest procés es repeteix fins assolir la convergència a una política òptima.

Avaluació de la política Un primer pas és trobar V^π per una política π fixada. Això es coneix com el *problema de predicció*. Per això usarem l'equació 2.2.3 de l'apartat anterior. Com coneixem el model del MDP, aquestes equacions formen un sistema de $|S|$ equacions de $|S|$ incògnites (els valors de V^π per cada $s \in S$). Aquest sistema es pot resoldre amb programació lineal o amb un procediment iteratiu transformant l'equació de Bellman en una regla d'actualització (mirant un pas més en el futur). D'una manera o altra, som capaços de resoldre el sistema, és a dir, trobar els valors de V^π.

Millora de la política Coneixent V^π, identificarem el valor de totes les accions, $Q^\pi(s, a)$, usant l'equació 2.2.6 amb $\max_{a' \in A} Q^*(s', a') = V^\pi(s')$ i la política π. Si $Q^\pi(s, a)$ és més gran que $V^\pi(s)$ per un $a \in A$ significa que podem millorar la política escollint una acció diferent, millor, en un estat particular. Per tant, avaluarem cada una de les accions en cada un dels estats i escollirem la millor acció en cada un dels estats basant-nos en el valor actual de V^π. Quan es doni el cas en que la política ja no pot millorar més significa que haurem trobat la política òptima que satisfà l'equació d'optimalitat de Bellman.

Value Iteration A diferència del mètode anterior, aquest mètode combina els dos passos, avaluació i millora de la política. L'avaluació de la política no és completa sinó que es trunca aquesta avaluació i es passa a millorar la política abans. De fet, combina immediatament el pas de millora de les polítiques amb les seves iteracions, centrant-se, així en l'estimació directa de la funció de valor. En essència, combina una versió truncada del pas d'avaluació amb el pas de millora que és, bàsicament, l'equació 2.2.4 transformada en una regla d'actualització. Finalment, el mètode genera una seqüència de funcions de valor V_i a la vegada que es produeixen els Q-valors. La política determinista π es pot calcular utilitzant l'equació 2.2.5. Així, l'algorisme comença amb una funció de valor arbitrària, s'itera actualitzant-ne els valors fins que la distància amb la següent aproximació és prou petita.

2.3.2 Reinforcement Learning: Solucions lliures de models

Els algorismes lliures de models s'identifiquen amb el nom *Reinforcement Learning o RL*. Aquests no confien en la disponibilitat d'un model perfecte del MDP sinó que es preocupen, principalment, d'obtenir una política òptima a base d'interactuar amb l'entorn. Això és una simulació de la política que, d'aquesta manera, genera mostres de transicions i recompenses en diferents estats. Aquestes mostres contenen informació de l'entorn i són utilitzades per estimar les funcions de valor acció-estat. Com el model és desconegut, cal que l'agent explori l'MDP per obtenir informació. Això indueix un compromís entre exploració i explotació que s'ha d'equilibrar per obtenir una política òptima.

En el context dels algorismes lliures de models podem diferenciar dos tipus de mètodes: RL *indirecte* i RL *directe*. El primer consisteix en aprendre les funcions de transició i

recompensa de l'entorn i, en el moment en què el model és suficientment correcte, aplicar un dels mètodes DP de la secció anterior. El segon consisteix en estimar els valors de les accions sense estimar el model del MDP. També és possible fer una combinació dels dos mètodes estimant els valors de les accions i utilitzar el model per accelerar l'aprenentatge.

Un altre aspecte a tenir en compte és què fer amb el problema *temporal credit assignment*, és a dir, determinar quines accions condueixen a un resultat determinat. És difícil avaluar la utilitat d'una acció quan els efecte d'aquesta es percebran més endavant. Una opció és esperar fins al final (de l'episodi) i castigar o premiar en funció del camí emprat. Tot i això, aquesta opció requereix molta memòria i es podria donar la situació de que no hi hagués final. En canvi, es poden usar mecanismes similars a la iteració de valors per ajustar el valor estimat d'un estat en funció de la recompensa immediata i del valor estimat (descomptat) del següent estat. Això es sol anomenar *temporal difference learning*[27] (aprenentatge de diferències temporals). La diferència principal amb els DP és que les funcions de transició i recompensa no poden aparèixer a les regles d'actualització. La classe d'algorismes que interactuen amb l'entorn i actualitzen les seves estimacions després de cada experiència es coneixen com *online* (en línia).

Exploració Per solucionar el problema exploració-explotació s'utilitza un mecanisme d'exploració el qual normalment pren la millor acció (explotació) però a vegades es pren una acció diferent (exploració). És a dir, és necessari explorar l'entorn per tal d'adquirir coneixement d'aquest i, a la vegada, explotar les accions que semblen òptimes. L'estratègia més bàsica és coneguda com la política ϵ-greedy la qual consisteix en que l'agent escull la millor acció amb una probabilitat $(1-\epsilon)$ o escull una acció aleatòria amb probabilitat ϵ, per un ϵ petit. Una altra força utilitzada és la coneguda amb el nom *Boltzmann*[28] (o *softmax*). L'estratègia de selecció d'accions segueix sent aleatòria però les probabilitats de selecció es ponderen pels seus Q-valors relatius.

Temporal Difference Learning (TD) Els algorismes aprenen estimacions de valors basades en altres estimacions (anomenat *bootstraping*). Cada pas en l'entorn genera un exemple d'aprenentatge que pot ser utilitzat per aportar cert valor d'acord amb la recompensa immediata i el valor estimat del següent estat o parella estat-acció. El major principi del TD és que no cal esperar fins al final per realitzar actualitzacions.

Aquests mètodes tenen avantatges respecte els DP: no requereixen del model del MDP, s'implementen online de manera que es poden usar en circumstàncies diverses o només s'actualitzen els camins experimentats després de cada pas.

TD(0)[27] És un algorisme de la família d'algorismes d'aprenentatge TD. Soluciona el problema d'estimació de V^π per una política π donada de manera online i incremental. El TD(0) es pot usar per avaluar una política i funciona amb l'ús de la següent regla d'aprenentatge:

$$V_{k+1}(s) := V_k(s) + \alpha\Big(r + \gamma V_k(s') - V_k(s)\Big) \tag{2.3.1}$$

on $\alpha \in [0,1]$ és el *learning rate* (taxa d'aprenentatge) que determina en quina quantitat els valors s'actualitzen. Aquesta actualització es realitza després d'experimentar la transició d'un estat s a s' per l'acció a rebent com a recompensa r.

Q-learning Un dels més mètodes bàsics i populars per estimar las funcions de Q-valor és l'algorisme Q-learning[29]. La idea fonamental és estimar, incrementalment, els Q-valors per les accions en funció de la recompensa rebuda i la funció de Q-valor. La regla d'actualització utilitza els Q-valors i un operador màxim sobre els Q-valors del següent

estat:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha\left(r_t + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)\right) \tag{2.3.2}$$

L'agent realitza un pas a l'entorn de l'estat s_t a s_{t+1} utilitzant l'acció a_t rebent la recompensa r_t actualitzant, d'aquesta manera, el Q-valor de l'acció a_t a l'estat s_t des del qual s'ha executat l'acció.

L'algorisme Q-learning es considera *off-policy*, és a dir, la política que avaluem i busquem millorar és diferent a la política que usem per seleccionar les accions.

SARSA És un exemple de mètode *on-policy* que aprèn la funció de Q-valor. SARSA[30], el qual significa *State-Action-Reward-State-Action*, utilitza la següent regla d'actualització:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha\left(r_t + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)\right) \tag{2.3.3}$$

on l'acció a_{t+1} correspon a l'acció executada per la política a l'estat s_{t+1}. Aquest algorisme és especialment útil en entorns no estacionaris ja que, en aquestes situacions, no s'assoleix mai la política òptima.

Actor-Critic Learning Un altre classe d'algorismes que precedeixen el Q-learning i el SARSA són els mètodes *Actor-Critic*[31] els quals aprenen on-policy, és a dir la política que s'avalua i es millora és la mateixa que la usada per seleccionar les accions. Aquesta classe de mètodes mantenen separada una política independent de la funció de valor. A la política se l'anomena l'*Actor* i a la funció de valor el *Critic*. El Critic avalua les accions executades per l'actor usant l'error TD:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{2.3.4}$$

L'objectiu d'aquest error és canviar la preferència d'escollir una acció a en un estat s, denotat per $p(s, a)$. Aquesta preferència pot ser modificada usant:

$$p(s_t, a_t) := p(s_t, a_t) + \beta\delta_t \tag{2.3.5}$$

on el paràmetre β indica la quantitat d'actualització. Un dels avantatges que presenta tenir separada la política és que, quan hi ha moltes accions o l'espai és continu, no cal considerar tots els Q-valors de les accions per seleccionar-ne una d'elles. A més, poden aprendre polítiques estocàstiques de forma natural.

Mètodes de Monte Carlo Un altre tipus d'algorismes són els mètodes de Monte Carlo[32] (MC). Aquest tipus de mètodes mantenen el recompte de freqüències de transicions i recompenses i basen els seus valors en aquestes estimacions. Els algorismes MC tracten la recompensa a llarg termini com una variable aleatòria i prenen com a estimació la mitjana de les mostres. En contrast amb els mètodes TD, els mètodes MC calculen els valors basats en la mitjana de retorns observats durant la interacció. Una manera d'usar els mètodes MC és en el pas d'avaluació de la política en el mètode d'iteració de polítiques o en l'estimació del valor d'una acció.

2.4 Deep Reinforcement Learning

El Deep Reinforcement Learning (aprenentatge per reforç profund) consisteix en combinar el Reinforcement Learning amb el Deep Learning (aprenentatge profund). Vegem doncs

en què consisteix el Deep Learning i, a continuació, com podem combinar aquestes dues disciplines.

2.4.1 Deep Learning

El Deep Learning és un tipus d'aprenentatge automàtic basat en xarxes neuronals artificials que busca extreure característiques d'unes dades per realitzar una tasca concreta. D'aquesta manera, definim un tipus de problemes que busquen realitzar una tasca en concret, per exemple reconeixement d'imatges o la traducció lingüística, a partir d'un conjunt de dades o a partir d'unes dades generades pel propi algorisme. Així doncs, podem representar aquest tipus de problemes com una funció $f : \mathcal{X} \times \mathbb{R}^{nW} \to \mathcal{Y}$ parametritzada per $W \in \mathbb{R}^{nW} (n_W \in \mathbb{N})$, definida com $y = f(x, W)$.

L'objectiu és ajustar els valors del paràmetre W per tal de realitzar una tasca en concret. Tal i com hem definit el Deep Learning, es fa ús d'una xarxa neuronal artificial (profunda). Aquestes consisteixen en una successió de múltiples *capes* de processament, les quals contenen un conjunt de nodes anomenats *neurones*. Cada una d'aquestes capes consisteix en una transformació no lineal. La seqüència d'aquestes transformacions dona lloc a l'aprenentatge de diferents nivells d'abstracció. Això es reflexa amb un conjunt de nodes per capa que rep l'entrada de la capa anterior i genera una sortida cap a la següent capa.

Vegem doncs un exemple simple d'una xarxa neuronal amb una capa oculta totalment connectada (veure la figura 2.1). La primera capa rep els valors d'entrada x com un vector de mida $n_x \in \mathbb{N}$ (a la figura, $n_x = 3$). Els valors de la capa oculta són una transformació d'aquests valors per una funció no lineal, això és una multiplicació de matrius per W_1 de mida $n_h \times n_x$ amb $n_h \in \mathbb{N}$ més un terme b_1 de mida n_h (a la figura $n_h = 5$), seguidament s'aplica una transformació no lineal A, la *funció d'activació*:

$$h = A(W_1 \cdot x + b_1) \tag{2.4.1}$$

La funció d'activació no lineal s'encarrega de que la transformació a cada capa deixi de ser lineal. En el nostre cas només tenim una capa oculta però en podria haver més realitzant les transformacions fins arribar a la capa de sortida que dona els valors de sortida y. En aquest cas:

$$y = A(W_2 \cdot h + b_2) \tag{2.4.2}$$

on W_2 té mida $n_y \times n_h$ i b_2 té mida $n_y \in \mathbb{N}$ (en el nostre cas $n_y = 2$).

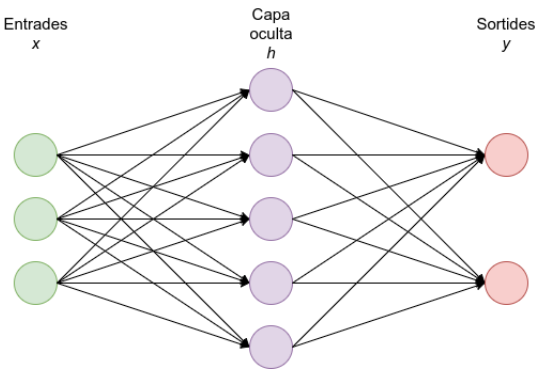

Figura 2.1: Exemple d'una xarxa neuronal amb una sola capa oculta

Fins ara, hem vist la transformació no lineal que realitza la funció f, a través de la xarxa neuronal, als valors d'entrada x produint una sortida $\hat{y} = f(x, W)$ que representa una predicció pels valors d'entrada donats.

Ens falta veure, doncs, com millorar aquestes prediccions, és a dir, com el mètode aprèn. Per veure això, cal construir una funció de de cost (o de pèrdua). Això es pot fer de diverses maneres ja que en funció del problema tindrem o no el valor real y. En el cas en que no, cada algorisme buscarà proporcionar un valor per tal de poder comparar-lo amb el valor predit i construir la funció de cost.

Un exemple de funció de cost és la *Mean Squared Error*, MSE (la mitjana d'error quadràtic). Suposant que tenim $n \in \mathbb{N}$ dades $\{(x_i, y_i)\}_{i=1}^n$ produint les prediccions $\{\hat{y}_i\}_{i=1}^n, \hat{y}_i = f(x_i, W)$, construïm la funció de cost MSE com:

$$C = \frac{1}{n} \sum_{k=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.4.3}$$

Ara, el nostre objectiu és minimitzar aquesta funció d'error actualitzant els valors dels paràmetres W de la funció $f(x, W)$, aquest mètode és conegut com a *Backpropagation*.

Ens trobem doncs en un problema d'optimització el qual podem resoldre amb l'algorisme *Gradient Descent*[33] (Descens del gradient). Un mètode iteratiu que actualitza els paràmetres W de la funció fins assolir el mínim (almenys local) de la funció de cost $C(W) = \frac{1}{n} \sum_{k=1}^{n} (y_i - f(x_i, W))^2$ amb les parelles $(x_i, y_i), i = 1, ..., n$ donades. Els paràmetres s'actualitzen en la direcció contrària al gradient seguint la següent regla d'actualització:

$$W_{k+1} = W_k - \alpha_k \nabla C(W_k) \tag{2.4.4}$$

on $k \in \mathbb{N}$ indica la k-èssima iteració.

D'aquesta manera, quan l'algorisme convergeixi quedaran els paràmetres actualitzats i podrem repetir el procés d'avaluació de la funció inicial amb un conjunt nou de dades o amb les mateixes però barrejades. D'aquesta manera ajustarem els paràmetres W de nou fins que arribi un moment que aquests siguin els òptims en termes de generalització.

Aquest primer procés, es sol anomenar procés d'entrenament o aprenentatge durant el qual la xarxa neuronal aprèn els valors dels paràmetres. Finalment, toca avaluar la xarxa neuronal amb un nou conjunt de dades el qual no ha sigut usat durant l'entrenament. D'aquesta manera, podem valorar en quina mesura el mètode utilitzat ha après a generalitzar la tasca objectiu. Per aquest motiu, abans de començar tot el procés, donat un conjunt de dades $\{(x_i, y_i)\}_{i=1}^n$, cal dividir aquest conjunt en dos subconjunts: el conjunt de dades d'entrenament i el conjunt de dades de test o avaluació, per exemple un 70% per a l'entrenament i un 30% per a l'avaluació.

2.4.2 Reinforcement Learning + Deep Learning

La incorporació del Deep Learning al Reinforcement Learning genera un nou tipus d'algorismes del Deep Learning, el Deep Reinforcement Learning. Aquest consisteix en incorporar l'estructura anterior, les xarxes neuronals, en els algorismes de Reinforcement Learning permetent resoldre una àmplia gamma de tasques complexes que, anteriorment, no estaven a l'abast d'una màquina.

Podem diferenciar dos tipus d'algorismes: els mètodes basats en valors, els quals optimitzen una funció de valor de la qual s'extreu la política, i els mètodes basats en polítiques,

els quals optimitzen directament la política. Vegem doncs un algorisme de cada tipus que ens permetrà exemplificar com s'incorpora el Deep Learning dins els algorismes de Reinforcement Learning.

Deep Q-Network[34] (DQN) És un exemple de mètode basat en valors, aquest és l'extensió de l'algorisme de Q-learning incorporant una xarxa neuronal, en particular una *Convolutional Neural Network (CNN)*[35] (xarxa neuronal convolucional), per aproximar la funció de valor d'acció òptima, és a dir, aproxima la funció de Q-valor. D'aquesta manera es substitueix la taula de Q-valors de l'algorisme Q-learning per una CNN. La CNN rep com entrada un estat i retorna, com a sortida, els Q-valors per cada una de les accions. Construïm la funció de cost com el MSE entre el Q-valor predit i el Q-valor objectiu. Això és, recuperant l'equació del Q-learning 2.3.2:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha\left(r_t + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)\right) \tag{2.4.5}$$

el Q-valor objectiu és $r_t + \gamma \max_a Q_t(s_{t+1}, a)$ i el Q-valor predit $Q_t(s, a)$ parametritzat per W, els paràmetres que cal ajustar per minimitzar la funció de cost.

Al combinar bootstraping, off-policy i aproximacions de funcions es pot produir certa inestabilitat i divergència a causa dels següents factors: correlacions entre observacions seqüencials, petites actualitzacions de la funció de Q-valors poden canviar dràsticament la política i, per últim, correlacions entre els Q-valors objectiu i predit.

Per solucionar-ho, el DQN utilitza *Experience Replay* i xarxes objectiu. L'experience replay consisteix en guardar les seqüències d'observacions (s_t, a_t, r_t, s_{t+1}) en un *Replay Buffer* i s'agafen aleatòriament per evitar les correlacions de les dades i suavitzar els canvis de distribució de dades. La xarxa objectiu manté els paràmetres de la xarxa separats actualitzant-los periòdicament per reduir les correlacions entre els Q-valors predits i els Q-valors objectiu.

Deep Deterministic Policy Gradient[36] (DDPG) És un exemple de mètode basat en polítiques, en particular pertany a un ampli subconjunt d'aquest tipus els quals s'anomenen *Policy Gradient Methods*. El DDPG és l'extensió de l'algorisme *Deterministic Policy Gradient*[37]. El DDPG està constituït per 4 xarxes neuronals: l'actor, l'actor objectiu, el crític i el crític objectiu. Els actors són els encarregats de la política, és a dir, aprendre la política òptima i determinar l'acció a realitzar en cada estat. Els crítics s'encarreguen de la funció de valor igual que amb l'algorisme DQN, és a dir, aprenen a calcular els Q-valors.

A més, el DDPG introdueix una representació directa de la política que li permet superar la restricció d'espais d'accions discretes. Als espais d'accions discretes denotem per $\pi(s)$ la política construint-la iterativament amb $\pi_{k+1}(s) = \arg\max_{a \in A} Q^{\pi_k}(s, a)$ a la k-èssima iteració. D'altra banda, en espais continus pot suposar un problema al requerir una maximització a cada pas. Denotem doncs $\pi_w(s)$ una política determinista diferenciable. En aquest cas, una alternativa simple i computacionalment atractiva és moure's en la direcció del gradient de Q que condueix a l'algorisme DDPG:

$$\nabla_w V^{\pi_w}(s_0) = \mathbb{E}_{s\ \rho^{\pi_w}}\left[\nabla_w(\pi_w)\nabla_a(Q^{\pi_w}(s, a))\big|_{a=\pi_w(s)}\right] \tag{2.4.6}$$

Capítol 3

Anàlisi del Problema

En aquesta secció veurem en detall en què consisteix el problema a tractar així com els elements que el conformen i la base des de la qual partim. Ens situem en el context del Reinforcement Learning, és a dir tal i com hem vist en la secció anterior tenim un entorn i un agent. Igual que en la majoria de problemes de machine learning, es busca proporcionar una solució que sigui capaç de generalitzar en la tasca concreta que està destinat a fer. Dins del context del Reinforcement Learning això consisteix en què el nostre agent aprengui una política, o una funció de la qual es pugui derivar aquesta política, que li permeti generalitzar el seu comportament a partir de l'experiència prèvia i, d'aquesta manera, pugui aconseguir bons resultats en entorns no vistos anteriorment.

Construïm doncs el nostre problema d'una manera més formal. Vegem primer una visió general del *Retro Contest*[14] i, seguidament, el *benchmark* (entorn de desenvolupament, o proves, de referència) basat en la franquícia de videojocs *Sonic the Hedgehog™* el qual ens definirà l'agent i l'entorn.

3.1 Retro Contest

Consisteix en un concurs presentat per l'empresa OpenAI l'any 2018. Juntament amb el concurs també es presenta el *Gym Retro*[14] una plataforma que permet utilitzar jocs clàssics com a entorns pel reinforcement learning. L'objectiu del concurs era mesurar les capacitats de *transfer learning* i *few-shot learning* dels algorismes DRL. D'una banda, el transfer learning consisteix en aplicar els coneixements obtinguts en els entorns d'entrenament a nous entorns que l'agent encara no ha explorat. D'altra banda, el few-shot learning consisteix en limitar l'aprenentatge de l'agent, en el nostre cas, limitant-ne el temps d'execució. L'objectiu del few-shot learning és construir models precisos reduint els costos temporals, computacionals, etc.

El concurs posava a disposició dels participants la plataforma Gym Retro juntament amb un benchmark basat en el *Sonic the Hedgehog™*. Com a afegit es proporcionaven una sèrie d'algorismes com a punt de partida els quals havien sigut avaluats en aquest nou benchmark (secció 4[14]).

3.2 Sonic Benchmark

El Sonic benchmark està dissenyat per mesurar el rendiment del transfer learning i del few-shot learning d'algorismes de reinforcement learning. Aquest ofereix divisions apropiades dels entorns en conjunts d'entrenament i d'avaluació fent-lo ideal per mesurar la *cross-task generalization*, és a dir és ideal per mesurar algorismes que busquen la generalització entre tasques similars.

El benchmark està dividit en diferents aspectes. Les següents subseccions mostren en detall cada un d'aquests aspectes.

3.2.1 Gym Retro

Sota el benchmark del Sonic està el Gym Retro, un projecte dirigit a crear entorns pel reinforcement learning a partir de l'emulació de diversos videojocs. Al nucli d'aquest està el paquet Python *gym-retro* el qual permet exposar jocs emulats com a *Gym environments*[38] utilitzant la *Libretro API*[1] per comunicar-se amb els emuladors dels jocs.

El gym-retro inclou un conjunt de jocs on cada un d'aquests consisteix en una *ROM* que conté les dades i codi del joc, un o més *estats guardats* els quals poden ser una captura de l'estat de la consola en un moment determinat, un *fitxer de dades* el qual descriu on es guarda determinada informació a la memòria de la consola i, per últim, un *escenari* el qual descriu les condicions que defineixen un estat final i la funció de recompensa.

3.2.2 El videojoc del Sonic

En aquest benchmark s'utilitzen 3 jocs similars: *Sonic The Hedgehog™*, *Sonic The Hedgehog™2* i *Sonic 3 & Knuckles*. Aquests tenen regles i controls similars per tant podem utilitzar múltiples jocs amb l'objectiu de generar tants entorns com sigui possible.

Cada joc està dividit en zones i cada zona està dividida en actes. Tot i que les regles i l'objectiu principal són els mateixos durant tot el joc, cada zona té un conjunt únic de textures i objectes. Diferents actes d'una zona poden compartir aquestes característiques però es diferencien per la posició espacial. D'aquesta manera ens referirem a la tripleta (ROM, zona, acte) com a nivell.

El benchmark consisteix en un total de 58 estats guardats obtinguts dels 3 jocs on cada un d'aquests estats conté el jugador a l'inici de cada nivell. Per tant, tenim un conjunt de 58 nivells el qual s'ha dividit en un conjunt d'entrenament i un d'avaluació. Aquesta divisió ve donada pel plantejament del concurs escollint aleatòriament zones amb més d'un acte i després seleccionant un dels actes a l'atzar de cadascuna d'aquestes zones. Els conjunts de nivells d'entrenament i avaluació es troben a les taules de l'Annex B.

3.2.3 Interacció Agent-Entorn

Els entorns del gym-retro produeix una observació al principi de cada pas. Aquesta és sempre una imatge RGB de 24 bits per píxel (8 bits per cada canal R, G i B). En el Sonic, aquesta imatge és de 320 píxels d'amplada i 224 píxels d'alçada la qual definirà l'estat

[1] https://www.libretro.com/index.php/api/

en un moment determinat. La Figura 3.1 mostra un estat per cada un dels jocs que hem mencionat anteriorment.

Figura 3.1: Captures de pantalla del Sonic. Cada una d'elles consisteix en una imatge de 320x224 píxels representant un estat. **Esquerra:** Estat inicial del SonicTheHedgehog2-Genesis EmeraldHillZone Act1. **Centre:** Estat inicial del SonicTheHedgehog-Genesis GreenHillZone Act1. **Dreta:** Estat inicial del SonicAndKnuckles3-Genesis AngelIslandZone Act1.

A cada pas, l'agent realitzarà una acció que representarà una combinació de botons de la consola. Pels jocs de la Sega els botons són els següents: *B, A, MODE, START, UP, DOWN, LEFT, RIGHT, C, Y, X, Z.* Pel Sonic podem considerar un subconjunt de combinacions més petit, aquest serà el següent: {{}, {*LEFT*}, {*RIGHT*}, {*LEFT, DOWN*}, {*RIGHT, DOWN*}, {*DOWN*}, {*DOWN, B*}, {*B*}}.

Com ens trobem en un entorn episòdic, és a dir existeixen estats finals, definim les condicions que condueixen a aquest estat final i que, a continuació, comportaran un restabliment de l'entorn al seu estat inicial. Aquestes condicions són: l'agent completa el nivell satisfactòriament que correspon a passar un determinat punt del nivell, l'agent perd una vida o l'agent assoleix el nombre màxim de passos definit com 4500.

La recompensa que rep l'agent consisteix en dues components: el desplaçament horitzontal més un bonus per completar el nivell. Els agents reben recompenses de manera que la recompensa acumulada en un punt determinat és proporcional al desplaçament horitzontal des del punt inicial. Aquesta mesura està normalitzada per nivell de tal manera que la recompensa total és 9000 si es completa el nivell satisfactòriament. El bonus per completar el nivell és de 1000 assolint el final del nivell automàticament i decreix linealment a zero als 4500 passos.

D'aquesta manera, podem definir el nostre problema com un Markov Decision Process com hem definit a la secció anterior 2.2. Recordem que un Markov Decision Process està definit per la tupla $\langle S, A, T, R \rangle$ on S és el conjunt d'estats, A el conjunt d'accions, T la funció de transició i R la funció de recompensa. Veiem doncs en que consisteixen cada un d'aquests elements tenint en compte el que hem vist en la descripció del benchmark del Sonic.

Tant el conjunt d'estat S com el conjunt d'accions A ve definit per l'entorn Gym de cada nivell. S és l'espai d'observacions, continu, on cada estat és una imatge RGB de mida 320x224 píxels. A és l'espai d'accions, discret, el qual hem definit com {{}, {*LEFT*}, {*RIGHT*}, {*LEFT, DOWN*}, {*RIGHT, DOWN*}, {*DOWN*}, {*DOWN, B*}, {*B*}}.

Les funcions de transició i recompensa també venen donades per l'entorn, però l'agent és aliè a la seva definició. La funció de transició està definida per l'entorn de manera que quan l'agent executa una acció en un estat determinat l'entorn retorna el nou estat.

La funció de recompensa també està definida per l'entorn, tot i que, com hem vist, el Gym Retro ens permet personalitzar-la, així doncs, aquesta es defineix tal com hem vist anteriorment.

Així doncs, tenim un problema pel qual volem que el nostre agent desenvolupi una política mitjançant una tècnica lliure de models, és a dir un algorisme de reinforcement learning que ens permeti desenvolupar aquesta política amb l'objectiu d'assolir la màxima recompensa sense disposar d'un model d'aquest entorn.

3.3 Anàlisi d'objectius

La finalitat d'aquesta secció és estudiar els objectius vists a la secció 1 per veure com es volen assolir aquests objectius. Vegem doncs quins són els objectius d'aquest treball i, a continuació, veurem en detall l'estratègia que seguirem en les següents seccions per tal d'assolir aquests objectius.

L'objectiu principal del treball és analitzar l'efecte que tenen els mètodes d'optimització en el rendiment d'algorismes de Deep Reinforcement Learning acotats al benchmark del Sonic. Aquest objectiu és molt ampli i el podem descompondre en objectius més específics: l'anàlisi i aplicació d'algorismes complexos de Deep Reinforcement Learning, l'estudi teòric de mètodes d'optimització i, per últim, l'anàlisi pràctic de l'aplicació dels algorismes d'optimització al Deep Reinforcement Learning. D'aquesta manera, per tal d'aconseguir l'objectiu principal, ens centrarem en assolir els objectius específics.

En primer lloc, ens centrarem en l'estudi teòric dels mètodes d'optimització. Plantejarem el problema d'optimització i veurem una classe d'algorismes basats en el Gradient Descent. Seguirem una línia d'evolució dels mètodes basats en el gradient descent que porten a un algorisme molt popular en l'àmbit del Deep Learning, l'*Adam*[21]. Analitzarem cada un d'aquests mostrant les millores aplicades a l'algorisme base, el gradient descent, centrant-nos especialment en l'Adam.

En segon lloc, veurem dos algorismes complexos de Deep Reinforcement Learning els quals han donat bons resultats en el benchmark del Sonic. Amb l'objectiu d'analitzar aquests algorismes veurem com es defineixen, les seves característiques principals i el procediment que hem seguit per tal d'avaluar-los en el Sonic.

Per últim, passarem a una fase d'experimentació en la qual aplicarem els algorismes de Deep Reinforcement Learning vistos anteriorment a la vegada que combinem aquests amb diferents algorismes d'optimització. D'aquesta manera, podrem avaluar el comportament d'ambdós tipus d'algorismes aplicats al benchmark del Sonic i comparar els rendiments d'aquests en funció del mètode d'optimització usat.

En resum, per tal d'assolir l'objectiu principal d'analitzar l'efecte dels algorismes d'optimització en el rendiment dels algorismes de Deep Reinforcement Learning aplicats al benchmark del Sonic realitzarem els seguents passos. Primer mostrarem l'estudi teòric de les dues classes d'algorismes, optimització i deep reinforcemnt learning. A continuació, aplicarem els algorismes en el benchmark del Sonic basant-nos en el coneixement obtingut. El fet de combinar els mètodes d'optimització amb els algorismes de deep reinforcement learning ens permetrà, a la fase d'avaluació dels resultats, analitzar l'efecte dels mètodes d'optimització en el rendiment dels algorismes de Deep Reinforcement Learning aplicats al benchmark del Sonic.

Capítol 4

Mètodes d'Optimització

Aquesta secció es centrarà en l'estudi teòric dels mètodes d'optimització introduint en primer lloc el problema d'optimització i, seguidament, en l'anàlisi de diferents mètodes i algorismes en el context del machine learning[39]. La finalitat d'aquesta secció és entendre el problema d'optimització que es presenta en algorismes basats en deep learning on cal minimitzar una funció de cost així com els mètodes que s'han desenvolupat per tal de resoldre'l de la manera més eficient possible.

Així doncs, podem observar que els algorismes de deep learning confien en gran mesura en l'eficiència dels algorismes d'optimització. Tal com hem vist a la secció 2.4.1, els optimitzadors juguen un paper important en el deep learning realitzant càlculs numèrics dels paràmetres d'un sistema dissenyat per prendre decisions. D'aquesta manera, partint d'un conjunt de dades d'entrada ajustar els paràmetres de la funció que defineix el nostre problema amb l'objectiu d'obtenir una sortida que s'ajusti al problema donat.

4.1 Introducció

A la secció 2.4.1 hem introduït el problema d'optimització que volem resoldre. És important, doncs, contextualitzar aquest aquest problema definint-lo de manera formal sense deixar de tenir en compte que les circumstàncies en les quals podem resoldre'l, és a dir, ordinadors amb memòria i capacitat de processament limitada.

4.1.1 Fonaments

L'objectiu del problema és determinar una funció de predicció $h : \mathcal{X} \to \mathcal{Y}$ d'un espai d'entrada $\mathcal{X}$ a un espai de sortida $\mathcal{Y}$ de tal manera que, donat $x \in \mathcal{X}$ el valor de $h(x)$ ofereix una predicció acurada del valor real de y. Per tant, estem buscant una funció que aprengui a generalitzar els conceptes que es poden aprendre d'un conjunt d'exemples donat, en lloc de memoritzar la relació entrada-sortida. Per assolir aquest objectiu, cal escollir una funció de predicció h intentant minimitzar el risc, una mesura que relaciona els valors $h(x)$ i y. Observem doncs que els problemes d'optimització estan determinats per les definicions de funcions de predicció i de pèrdua.

Funcions de predicció i pèrdua Suposem que la funció de predicció h té una forma fixa i està parametritzada per un vector $w \in \mathbb{R}^d$ sobre la qual s'ha de realitzar la optimització.

Formalment, per alguna $h(\cdot;\cdot) : \mathbb{R}^{d_x} \times \mathbb{R}^d \to \mathbb{R}^{d_y}$, considerem la família de funcions de predicció

$$\mathcal{H} := \{h(\cdot; w) : w \in \mathbb{R}^d\}$$

Volem trobar la funció de predicció d'aquesta família que minimitzi les pèrdues derivades de prediccions inexactes. Per això, suposem una funció de pèrdua determinada $\ell : \mathbb{R}^{d_y} \times \mathbb{R}^{d_y} \to \mathbb{R}$ de tal manera que donada una parella entrada-sortida $(x, y) \in \mathbb{R}^{d_x} \times \mathbb{R}^{d_y}$ produeix el valor pèrdua $\ell(h(x; w), y)$ on $h(x; w)$ i y són el valor predit i real respectivament.

Definim ara dos conceptes basats en aquesta funció de pèrdua els quals definiran, de manera genèrica, una funció de cost que representarà l'error comès entre el valor real i el predit.

Risc esperat Idealment, el vector de paràmetres w s'escull per minimitzar la pèrdua esperada produïda per qualsevol parella d'entrada-sortida. Formalment, suposem que les pèrdues es mesuren respecte una distribució de probabilitats $P(x, y)$ que representa la relació real entre entrades i sortides. És a dir, suposem que l'espai d'entrada-sortida $\mathbb{R}^{d_x} \times \mathbb{R}^{d_y}$ està dotat de $P : \mathbb{R}^{d_x} \times \mathbb{R}^{d_y} \to [0, 1]$ i la funció que volem minimitzar és:

$$R(w) = \int_{\mathbb{R}^{d_x} \times \mathbb{R}^{d_y}} \ell(h(x; w), y)\, dP(x, y) = \mathbb{E}[\ell(h(x; w), y)] \tag{4.1.1}$$

Diem que $R : \mathbb{R}^d \to \mathbb{R}$ produeix el *risc esperat* (és a dir, la pèrdua esperada) donat un vector de paràmetres w respecte la distribució de probabilitats P.

Risc empíric Tot i que seria desitjable minimitzar 4.1.1, aquest objectiu no és factible quan no es té completament la informació de P. Per aquest motiu, a la pràctica, es busca la solució d'un problema que implica l'estimació del risc esperat R. Si tenim en compte que el problema es troba dins del context del deep learning, podem suposar que disposem d'un conjunt de $n \in \mathbb{N}$ finit de parelles entrada-sortida $(x_i, y_i)_{i=1}^n \subseteq \mathbb{R}^{d_x} \times \mathbb{R}^{d_y}$ independents idènticament distribuïdes que ens permeten reduir l'expressió 4.1.1 al cas discret definint la funció *risc empíric* $R_n : \mathbb{R}^d \to \mathbb{R}$ com:

$$R_n = \frac{1}{n} \sum_{i=1}^{n} \ell(h(x_i; w), y_i) \tag{4.1.2}$$

Així doncs, en els problemes d'optimització, idealment, busquem minimitzar el risc esperat definit a 4.1.1 però, a la pràctica, treballem amb l'aproximació R_n de 4.1.2. D'aquesta manera, el problema d'optimització correspon a minimitzar aquesta expressió, és a dir, trobar el vector de paràmetres w que, donat un conjunt de parelles entrada-sortida $(x_i, y_i)_{i=1}^n \subseteq \mathbb{R}^{d_x} \times \mathbb{R}^{d_y}$, minimitza la funció R_n.

4.1.2 Gradient Descent

El *Gradient Descent*[22], introduït a la secció 2.4.1 és un dels algorismes més populars per realitzar l'optimització i, per tant, la forma més comuna d'optimitzar les xarxes neuronals. Aquest algorisme va ser proposat inicialment per *M.A. Cauchy* el 1847 en el qual proposa utilitzar el gradient per resoldre una equació no lineal de la forma $f(x_1, x_2, ..., x_n) = 0$, on f és una funció contínua, de valors reals que mai esdevé negativa i que es manté contínua, almenys dins de certs límits. Aquest mètode, també conegut com *Steepest Descent*[33], es defineix de la següent manera:

Suposem que volem trobar el mínim d'una funció $f : \mathbb{R}^n \to \mathbb{R}$. Denotem el gradient de f com $g_k = g(x_k) = \nabla f(x_k)$. La idea general és calcular un pas cap a una determinada direcció, d_k, per exemple,

$$x_{k+1} = x_k + \alpha d_k, k = 0, 1, ..., \tag{4.1.3}$$

amb $\alpha \in \mathbb{R}^+$ prou petit, al qual anomenarem *learning rate* (taxa d'aprenentatge), i la direcció d_k ve donada pel gradient a la k-èssima iteració $d_k = -\nabla f(x_k)$.

Observem que la mida del pas α a cada iteració pot variar entre diferents iteracions, per exemple agafant $\alpha_k = \arg\min_{\alpha} f(x_k + \alpha d_k)$.

Així doncs, iterant l'expressió 4.1.3 obtindrem la successió $\{x_k\}_{k\geq 0}$ la qual convergeix a un punt x que minimitza la funció f. D'aquesta manera, podem dir que aquest punt x és un mínim de f i, en el cas que la funció sigui convexa, podem assegurar que es tracta del mínim absolut de f.

Vegem doncs que l'afirmació anterior és certa. Abans de tot, cal veure una sèrie de definicions i propietats que ens permetran demostrar la convergència de l'algorisme.

Definició 1. *Sigui $f : \mathbb{R}^d \to \mathbb{R}$ direm que és una funció convexa si $\forall x, y \in \mathbb{R}^d$ i $\lambda \in [0, 1]$ es compleix:*

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda) f(y). \tag{4.1.4}$$

Definició 2. *Sigui $f : \mathbb{R}^d \to \mathbb{R}$ direm que és Lipschitz si existeix una constant $L > 0$ tal que $\forall x, y \in \mathbb{R}^d$ es compleix:*

$$||f(x) - f(y)||_2 \leq L||x - y||_2. \tag{4.1.5}$$

La següent afirmació mostra el fet que el pla tangent a una funció convexa viu sota la corba:

Afirmació 1. *Sigui $f : \mathbb{R}^d \to \mathbb{R}$ convexa i diferenciable. Aleshores, $\forall x, y \in \mathbb{R}^d$,*

$$f(x) + \nabla f(x)^T (y - x) \leq f(y) \tag{4.1.6}$$

La següent afirmació complementa la propietat anterior per funcions convexes amb gradients Lipschitz.

Afirmació 2. *Sigui $f : \mathbb{R}^d \to \mathbb{R}$ convexa i diferenciable, tal que el seu gradient és Lipschitz, $L > 0$. Aleshores $\forall x, y \in \mathbb{R}^d$,*

$$f(y) \leq f(x) + \nabla f(x)^T (y - x) + \frac{L}{2}||y - x||_2^2 \tag{4.1.7}$$

Per últim, també necessitarem la següent propietat de les funcions convexes.

Afirmació 3. *Per tota funció convexa $f : \mathbb{R}^d \to \mathbb{R}$ i $y_1, \ldots, y_k \in \mathbb{R}^d$,*

$$f\Big(\frac{y_1 + \ldots + y_k}{k}\Big) \leq \frac{f(y_1) + \ldots + f(y_k)}{k} \tag{4.1.8}$$

El següent teorema mostra com els iterats del Gradient Descent convergeixen al mínim absolut per funcions convexes amb gradients Lipschitz.

Teorema. *Sigui $f : \mathbb{R}^d \to \mathbb{R}$ una funció convexa i diferenciable, tal que el seu gradient és Lipschitz, $L > 0$, i $x^* = \arg\min_x f(x)$. Aleshores, els iterats de l'algorisme Gradient Descent amb mida de pas $\alpha \leq 1/L$ satisfan el següent:*

$$f(x_k) - f(x^*) \leq \frac{||x_0 - x^*||_2^2}{2\alpha k} \tag{4.1.9}$$

Demostració. *Veure la demostració a l'Annex C.*

4.2 Variants del gradient descent

Podem distingir 3 variants del gradient descent[40] en funció de la quantitat de dades utilitzades per avaluar la funció de cost i, en conseqüència, per calcular-ne el seu gradient.

Considerem la funció de cost a minimitzar $J : \mathbb{R}^d \to \mathbb{R}$, parametritzada per $\theta \in \mathbb{R}^d$ i definirem η com el learning rate. Assumim que disposem d'n parelles entrada-sortida $\Omega := \{(x_i, y_i) \in \mathbb{R}^{d_x} \times \mathbb{R}^{d_y}; i = 1, ..., n\}$. Com hem dit, aquests mètodes es diferencien en el conjunt de parelles entrada-sortida utilitzades per definir la funció de cost. El gradient descent utilitzava el conjunt complet Ω a cada iteració però podríem usar només un subconjunt d'aquest buscant reduir el temps emprat per realitzar una actualització a canvi de reduir la precisió amb la qual s'actualitzen aquests paràmetres.

En cada una de les variants veurem l'actualització que es realitza sobre els paràmetres a cada iteració. Definim el concepte *epoch* com el conjunt d'iteracions per visitar totes les dades del conjunt. Iterarem doncs l'algorisme realitzant una o més actualitzacions dels paràmetres per epoch fins que l'algorisme convergeixi al mínim buscat.

4.2.1 Batch Gradient Descent

El *Batch Gradient Descent* (BGD), també conegut com Vanilla gradient descent o full gradient descent, consisteix en l'algorisme bàsic descrit anteriorment. Es calcula el gradient de la funció objectiu J respecte els paràmetres θ per a tot el conjunt de dades Ω. Així, la regla d'actualització dels paràmetres és la següent:

$$\theta_{k+1} = \theta_k - \eta \cdot \nabla_{\theta_k} J(\theta_k) \tag{4.2.1}$$

Com cal calcular els gradients per tot el conjunt de dades per realitzar només una actualització, el batch gradient descent pot arribar a ser molt lent i és intractable per conjunts de dades que no caben en memòria. A més, tampoc permet actualitzar el model *online*, és a dir a mesura que apareixen noves parelles. Tot i això, podem garantir que el batch gradient descent convergeix a un mínim local o global de J per l'optimització no convexa i convexa respectivament.

4.2.2 Stochastic Gradient Descent

En contrast, el *Stochastic Gradient Descent* (SGD) utilitza només un element del conjunt Ω, $(x^{(i)}, y^{(i)})$ a cada iteració. D'aquesta manera, els paràmetres s'actualitzen n vegades per epoch utilitzant únicament una parella entrada-sortida a cada iteració:

$$\theta_{k+1} = \theta_k - \eta \cdot \nabla_{\theta_k} J(\theta_k; x^{(i)}; y^{(i)}) \tag{4.2.2}$$

Observem que el BGD realitza càlculs redundants per a conjunts de dades grans, ja que calcula gradients per a parelles $(x^{(i)}, y^{(i)})$ similars diverses vegades abans d'actualitzar els paràmetres. El SGD el·limina aquesta redundància realitzant una actualització per cada parella. Per aquest motiu, el SGD sol ser més ràpid i permet l'aprenentatge online. Tot i això, degut a l'alta freqüència d'actualització dels paràmetres, la funció de cost J fluctua fortament. D'una banda, aquesta fluctuació li permet saltar a mínims potencialment millors, d'altra banda, això complica la convergència al mínim de manera exacte. Tot i això, s'ha demostrat que reduint la taxa d'aprenentatge gradualment, l'algorisme té el mateix tipus de convergència que el BGD, és a dir, convergeix a un mínim local o global per l'optimització no convexa i convexa respectivament[41]. Nota: en cada epoch, un cop explorades totes les dades del conjunt Ω, aquestes es barregen per evitar que l'algorisme utilitzi les dades en el mateix ordre esbiaixant l'algorisme d'optimització.

4.2.3 Mini-Batch Gradient Descent

El *Mini-Batch Gradient Descent* (MBGD) combina les dues variants anteriors amb l'objectiu d'integrar les millors característiques de cada una d'elles. El MBGD funciona de manera semblant al SGD però, en lloc d'utilitzar una única parella $(x^{(i)}, y^{(i)})$ per iteració, aquest utilitza $m \in \mathbb{N}, m < n$ parelles de Ω, al qual anomenarem *mini-batch*, per avaluar el gradient de J i actualitzar els paràmetres:

$$\theta_{k+1} = \theta_k - \eta \cdot \nabla_{\theta_k} J(\theta_k; x^{(i:i+n)}; y^{(i:i+n)}) \tag{4.2.3}$$

Observem doncs que, a cada epoch, el MBGD realitza $\approx \frac{n}{m}$ iteracions actualitzant els paràmetres. Reduint, d'aquesta manera, la variació d'actualització dels paràmetres, evitant la fluctuació de la funció i conduint-la a una convergència més estable. A més, es pot fer ús de llibreries altament optimitzades en l'optimització de matrius, les quals fan que calcular el gradient respecte un mini-batch sigui molt eficient. Normalment, els mini-batch tenen mides d'entre 50 i 256 exemples. Nota: igual que abans, a cada epoch, es barregen les dades per evitar esbiaixar l'algorisme.

4.3 Motivació pels mètodes Estocàstics

Fins ara hem vist tres variants del gradient descent. A l'annex C podem veure l'anàlisi de la convergència del BGD, equivalent al gradient descent bàsic, i del SGD amb mides de pas fixes. Observem que tenen ratis de convergència semblants, $\mathcal{O}(1/k)$ i $\mathcal{O}(1/k) + \mathcal{O}(\alpha)$ respectivament. Així doncs, ens podem preguntar per què no utilitzar sempre el BGD ja que, matemàticament, dona els millors resultats gràcies a la precisió del càlcul del gradient al utilitzar tot el conjunt de dades Ω. I, a més, el BGD presenta els següents avantatges.

En primer lloc, un cop hem reduït el problema a minimitzar el risc empíric R_n, el fet d'utilitzar tota la informació del gradient a cada iteració obre les portes a una gran diversitat de mètodes deterministes d'optimització basada en gradients. És a dir, es tenen a disposició diferents tècniques d'optimització no lineal, inclòs el full gradient descent, però també accelerated gradient, conjugate gradient, quasi-Newton i mètodes de Newton inexactes.

En segon lloc, degut a l'estructura de la suma R_n, el mètode batch es pot beneficiar de la paral·lelització ja que la major part del càlcul rau en avaluacions de R_n i ∇R_n. Fins i tot es poden fer càlculs d'aquestes quantitats d'una manera distribuïda.

Tot i aquestes avantatges, veurem una sèrie de raons[39], intuïtives, pràctiques i teòriques que justifiquen l'ús de mètodes estocàstics. Vegem-los contrastant les iteracions dels dos mètodes.

Raó intuïtiva Intuïtivament, hom pot pensar que el mètode estocàstic utilitza la informació més eficientment que el mètode batch. Per veure-ho, considerem la situació en què el conjunt d'entrenament, diguem-li $\mathcal{S}$, consisteix en deu còpies del conjunt $\mathcal{S}_{sub}$. Un minimitzador del risc empíric pel conjunt $\mathcal{S}$ és clarament donat per un minimitzador pel conjunt $\mathcal{S}_{sub}$. Si s'aplica el mètode batch per minimitzar R_n sobre el conjunt $\mathcal{S}$, aleshores, cada iteració del mètode serà deu vegades més costosa que si només es tingués una còpia de $\mathcal{S}_{sub}$. En canvi, el mètode estocàstic farà els mateixos càlculs en els dos escenaris ja que aquests càlculs involucren el fet d'escollir elements de $\mathcal{S}_{sub}$ amb la mateixa probabilitat. Realment, un conjunt d'entrenament, difícilment contindrà dades repetides però en el context d'una aplicació que utilitza una gran quantitat de dades, aquestes tenen altes probabilitats de ser aproximadament redundants.

Raó pràctica Els beneficis intuïts anteriorment s'han observat repetidament a la pràctica on sovint es troben grans avantatges en l'ús de mètodes estocàstics. Com a exemple, podem prendre la comparació del comportament d'un mètode batch, el L-BFGS[42], i el SGD anterior prenent α constant en un problema de classificació binari. Podem veure a la figura 4.1[39] la representació del risc empíric R_n en funció del nombre d'accessos de cada dada del conjunt d'entrenament, és a dir, el nombre d'avaluacions del gradient $\nabla f_i(w_k)$. El mètode batch fa un únic pas per epoch mentre que el SGD realitza n passos per epoch. La gràfica mostra el comportament en les 10 primeres epochs. Podem veure que el mètode SGD es comporta millor en termes de reducció del risc empíric en funció dels accessos a memòria. Tot i això, cal tenir en compte que per trobar el valor α cal executar repetidament el mètode amb diferents valors de α fins identificar una bona elecció pel problema en particular. També cal tenir en compte que si la funció estigués composada de funcions convexes el mètode pot tenir un comportament oscil·latori que disminuirà el progrés significativament, caldria la informació completa del gradient per tal de solucionar-ho. D'altra banda, utilitzant una seqüència de passos decreixent també podrem superar aquest problema oscil·latori garantint la convergència del mètode.

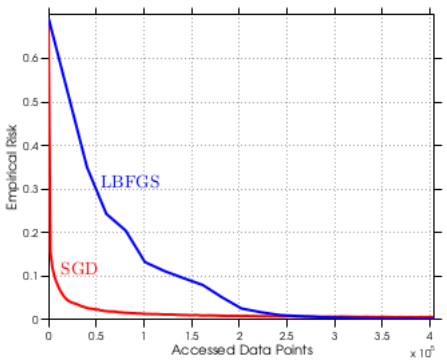

Figura 4.1: Imatge extreta de l'article [39] mostra la comparació, entre el mètode L-BFGS i un mètode SGD, del risc empíric en funció del nombre d'accessos en un problema de classificació binari.

Raó teòrica En aquesta raó, considerarem que R_n és fortament convexa (strongly convex), és a dir, $\forall x, y; R_n(y) \geq R_n(x) + \nabla R_n(x)^T(y-x) + \frac{\mu}{2}||y-x||^2$ per un cert $\mu > 0$.

Vegem doncs com es comporten, teòricament, els dos mètodes per tal de resoldre el mateix problema d'optimització.

Sabem que per la suposició 4.5 de [39], si apliquem un mètode batch existeix una constant $\rho \in (0,1)$ tal que per tot $k \in \mathbb{N}$, l'error de l'entrenament satisfà:

$$R_n(w_k) - R_n^* \leq \mathcal{O}(\rho^k) \tag{4.3.1}$$

on R_n^* denota el valor mínim de R_n. Aquest rati de convergència és R-lineal que, per simplificar, ens referirem a aquest com *convergència lineal*. Podem concloure que en el pitjor dels casos, el nombre total d'iteracions en el qual l'error d'entrenament pot ser major que un cert $\epsilon > 0$ és proporcional a $log(1/\epsilon)$. Tenint en compte que cada iteració té un cost n a causa d'haver de calcular $\nabla R_n(w_k)$ per cada $k \in \mathbb{N}$, podem concloure que el cost total per aconseguir una taxa de convergència ϵ-òptima per un mètode batch és proporcional a $nlog(1/\epsilon)$.
Ara, apliquem el mètode SGD tenint en compte que cada i_k s'escull de manera uniforme de $1, ..., n$. Pel Teorema 4.7 de [39] satisfà:

$$\mathbb{E}[R(w_k) - R^*] = \mathcal{O}(1/k) \tag{4.3.2}$$

D'aquesta manera, podem veure que el mètode SGD té una taxa de convergència més lenta que el mètode batch, tot i això, tant el cost per iteració com el costat dret de la igualtat 4.3.2 no depenen de la mida del conjunt, n. Això significa que, en aquest cas, el cost total necessari per aconseguir una taxa de convergència ϵ-òptima és proporcional a $1/\epsilon$. És cert que $1/\epsilon$ pot ser més gran que $nlog(1/\epsilon)$ per valors moderats de n i ϵ, però tal i com es mostra a l'apartat 4.4 de [39], aquesta comparació afavoreix al SGD quan passem al context del *big data* on n és molt gran i estem limitats pel temps computacional.

En resum, tenim raons intuïtives, pràctiques i teòriques a favor dels mètodes estocàstics sobre mètodes batch en problemes d'optimització de gran escala. Això no significa que els batch no siguin útils a la pràctica. D'una banda, si la figura 4.1 considerés un nombre més gran d'epochs, podríem veure que el mètode batch superaria el mètode estocàstic i produiria un error d'entrenament inferior. Per aquest motiu, això motiva a combinar els dos mètodes intentant combinar les millors propietats dels algorismes estocàstics i batch, donant lloc al mètode mini-batch presentat anteriorment.

4.4 Limitacions

Assumint un mètode estocàstic, sigui el SGD o el MBGD, no podem assegurar una bona convergència tenint en compte que, en el context de les xarxes neuronals, ens trobem amb problemes d'optimització no convexa on la funció a minimitzar pot tenir extrems locals o punts de sella. A més, les limitacions d'aquests algorismes fan sorgir una sèrie de reptes que cal abordar amb l'objectiu d'augmentar la velocitat de convergència i evitar convergir a extrems diferents de l'absolut. Aquests reptes són els següents:

Elecció del learning rate Una tasca difícil ja que un learning rate massa petit causa convergència lenta mentre que un learning rate massa alt pot dificultar la convergència fent que la funció fluctuï al voltant del mínim o fins i tot fent que divergeixi.

Addicionalment, el mateix learning rate s'aplica a totes les actualitzacions dels paràmetres. Si les dades són escasses potser ens interessa actualitzar les dades en diferent mesura.

Punts subòptims En funcions no convexes l'algorisme pot quedar atrapat en un mínim local però la dificultat sorgeix sobretot en els punts de sella, és a dir, punts on diferents dimensions tenen pendents contraris.

Gradients extrems En el sentit que aquests gradients desapareixen o exploten. En funció de l'arquitectura de la xarxa, el gradient dels paràmetres pot esdevenir una sèrie que vagi a infinit o a zero ràpidament.

4.5 Evolució dels algorismes

En aquesta secció veurem una sèrie d'algorismes àmpliament utilitzats en el deep learning. Aquests algorismes són un petit conjunt dels algorismes d'optimització existents. Els algorismes d'aquesta secció busquen solucionar o, almenys, donar millores respecte l'algorisme original, el gradient descent, als reptes presentats anteriorment. Recordem l'equació 4.2.1 de les seccions anteriors, l'actualització dels paràmetres del gradient descent:

$$\theta_{k+1} = \theta_k - \eta \nabla_{\theta_k} J(\theta_k) \tag{4.5.1}$$

on $\theta = (\theta_1, ..., \theta_m) \in \mathbb{R}^m$ i $k \in \mathbb{N}$ indica la k-èssima iteració.

4.5.1 Momentum

L'algorisme *Momentum*[17] busca augmentar la velocitat de convergència incloent un terme *momentum* a l'equació d'actualització dels paràmetres del gradient descent 4.5.1:

$$\begin{aligned} v_k &= \eta \nabla_{\theta_k} J(\theta_k) + \gamma v_{k-1} \\ \theta_{k+1} &= \theta_k - v_k \end{aligned} \tag{4.5.2}$$

on γ és el paràmetre momentum normalment amb valor 0.9 o similar. D'aquesta manera, la modificació del paràmetre a la iteració k depèn tant del gradient actual com de l'actualització anterior multiplicada pel factor momentum.

Aquest mètode ajuda a accelerar el SGD en la direcció rellevant afegint el terme momentum ja que al acumular els gradients anteriors s'augmenta el pes de les dimensions les quals tenen gradients apuntant en la mateixa direcció i es redueix el pes de les dimensions amb gradients contraris. Això, redueix les oscil·lacions del SGD dirigint-lo a més velocitat cap al mínim.

4.5.2 Nesterov Accelerated Gradient

L'algorisme *Nesterov Accelerated Gradient*[18] igual que l'algorisme anterior busca augmentar la velocitat de convergència afegint un terme momentum. Afegit a això, aquest busca reduir el pes de l'actualització abans de que el gradient canviï de direcció. Així, l'equació d'actualització queda:

$$\begin{aligned} v_k &= \eta \nabla_{\theta_k} J(\theta_k - \gamma v_{k-1}) + \gamma v_{k-1} \\ \theta_{k+1} &= \theta_k - v_k \end{aligned} \tag{4.5.3}$$

De nou, el valor del terme momentum γ sol ser 0.9. Igual que abans la modificació del paràmetre depèn de l'actualització anterior juntament amb el gradient però en aquest cas avaluat en $\theta_k - \gamma v_{k-1}$. Al calcular $\theta_k - \gamma v_{k-1}$ ens permet donar una aproximació de la següent posició dels paràmetres (només falta el gradient per tenir l'actualització completa). D'aquesta manera, deixem de calcular el gradient respecte els paràmetres actuals per calcular-lo respecte una aproximació del pròxim valor d'aquests aconseguint l'objectiu de reduir el pes de l'actualització si el gradient canvia de direcció.

4.5.3 AdaGrad

L'algorisme *AdaGrad*[19] a diferència dels algorismes anteriors busca adaptar el learning rate als paràmetres, realitzant actualitzacions majors o menors en funció de la freqüència dels paràmetres.

Anteriorment, hem actualitzat tots els paràmetres $\theta \in \mathbb{R}^m$ a la vegada ja que aquests utilitzen el mateix learning rate η. En canvi, com l'AdaGrad utilitza diferents learning rates per cada un dels paràmetres $\theta_i, i \in \{1, ..., m\}$ a cada iteració k veurem primer com s'actualitza cada paràmetre per separat i, seguidament, passarem a notació vectorial.

Per simplificar la notació, definim:

$$g_{k,i} = \nabla_{\theta_k} J(\theta_{k,i}) \tag{4.5.4}$$

Així doncs, per cada $i \in \{1, ..., m\}$ l'equació d'actualització del paràmetre θ_i a la k-èssima iteració és el següent:

$$\theta_{k+1,i} = \theta_{k,i} - \frac{\eta}{\sqrt{G_{k,ii} + \epsilon}} \cdot g_{k,i} \tag{4.5.5}$$

on $G_k \in \mathbb{R}^{m \times m}$ és una matriu diagonal de manera que cada element i, i és la suma dels quadrats dels gradients respecte θ_i fins la iteració k, mentre que $\epsilon > 0$ és un terme prou petit per tal d'evitar la divisió entre 0 (normalment d'ordre $1e^{-8}$).

Ara, per passar a notació vectorial només cal tenir en compte que el producte de G_t amb g_t és un producte de matrius, així:

$$\theta_{k+1} = \theta_k - \frac{\eta}{\sqrt{G_k + \epsilon}} \cdot g_k \tag{4.5.6}$$

Amb aquesta regla d'actualització, l'algorisme AdaGrad aconsegueix actualitzar cada un dels paràmetres amb pesos diferents que depenen de la freqüència amb què s'han actualitzat els paràmetres. Això s'aconsegueix amb la matriu G_t que manté la suma dels quadrats dels gradients de les iteracions anteriors disminuint el pes de l'actualització dels paràmetres actualitzats més freqüentment i augmentant-ne el dels paràmetres menys freqüents.

El principal benefici que ofereix l'AdaGrad és que el·limina la necessitat d'establir manualment el learning rate. Tot i això, la regla d'actualització d'aquest té un defecte, l'acumulació dels quadrats dels gradients al denominador. Com aquest valor sempre és positiu creixerà a cada iteració durant l'entrenament. D'aquesta manera el factor que multiplica el gradient es farà indefinidament petit i arribarà un moment en que l'algorisme no podrà adquirir nous coneixements.

4.5.4 Adadelta i RMSProp

L'algorisme *Adadelta*[20] és una extensió de l'AdaGrad que busca solucionar el seu defecte mantenint els gradients anteriors fins una mida fixa w. To i això, en lloc de guardar els w últims gradients, la suma dels gradients es defineix recursivament com una mitjana decreixent de tots els gradients anteriors. Així, aquesta mitjana a la iteració k es defineix com:

$$E[g^2]_k = \gamma E[g^2]_{k-1} + (1-\gamma) g_k^2 \tag{4.5.7}$$

on γ té un valor similar al momentum, vora 0.9.

Ara, només cal substituir la matriu G_k per $E[g^2]_k$ a l'equació d'actualització de l'AdaGrad 4.5.6. Reescrivint l'equació queda:

$$\begin{aligned} \Delta\theta_k &= -\frac{\eta}{\sqrt{E[g^2]_k + \epsilon}} g_k \\ \theta_{k+1} &= \theta_k + \Delta\theta_k \end{aligned} \tag{4.5.8}$$

on igual que amb l'AdaGrad, ϵ és un terme prou petit per tal d'evitar la divisió entre 0.

Observem que el denominador de la primera igualtat és exactament l'arrel de l'error quadràtic mitjà del gradient (*root mean squared (RMS) error*) per tant el podem substituir per l'abreviatura $RMS[g]_k = \sqrt{E[g^2]_k + \epsilon}$.

Per acabar, amb l'objectiu de mantenir les mateixes unitats en aquesta actualització, definim una altra mitjana decreixent, en aquest cas dels quadrats de les actualitzacions dels paràmetres:

$$E[\Delta\theta^2]_k = \gamma E[\Delta\theta^2]_{k-1} + (1-\gamma)\Delta\theta_k^2 \tag{4.5.9}$$

Igual que abans, el RMS de les actualitzacions dels paràmetres és $RMS[\Delta\theta]_k = \sqrt{\Delta\theta_k^2 + \epsilon}$ però aquest valor és desconegut a la k-èssima iteració així que l'aproximem amb el de la iteració anterior $RMS[\Delta\theta^2]_{k-1}$. Ara només queda substituir el learning rate η i ens queda la regla d'actualització de l'Adadelta:

$$\theta_{k+1} = \theta_k - \frac{RMS[\Delta\theta]_{k-1}}{RMS[g]_k} g_k \tag{4.5.10}$$

A més, d'aquesta manera ja no cal especificar un learning rate γ ja que l'hem el·liminat de l'equació.

Per acabar, l'algorisme *RMSProp*[43] també consisteix en una extensió de l'algorisme AdaGrad dissenyat amb l'objectiu de resoldre el seu defecte.

L'RMSProp i l'Adadelta es van desenvolupar a la vegada i de manera independent però, sorprenentment, van arribar a una solució pràcticament idèntica. Conceptualment, l'RMSProp no canvia respecte a l'Adadelta, de tal manera que la regla d'actualització de l'RMSProp és idèntica a la primera part de l'Adadelta 4.5.8:

$$\begin{aligned} E[g^2]_k &= \gamma E[g^2]_{k-1} + (1-\gamma) g_k^2 \\ \theta_{k+1} &= \theta_k - \frac{\eta}{\sqrt{E[g^2]_k + \epsilon}} g_k \end{aligned} \tag{4.5.11}$$

on es suggereixen els valors 0.9 pel paràmetre γ i 0.001 per η.

4.6 Adam

L'Adam[21] és un algorisme d'optimització basat en gradients de primer ordre de la funció objectiu que realitza estimacions adaptatives de moments d'ordre inferior. De manera genèrica, l'Adam és una extensió del SGD que combina les avantatges dels algorismes, també extensions del SGD, AdaGrad, que funciona bé amb gradients escassos, i RMSProp, que funciona bé en configuracions online i no estacionàries. A més, l'Adam inclou la tècnica *initialization bias correction*[21] (correcció del biaix d'inicialització) que li permet aconseguir millors estimacions en les primeres iteracions.

L'algorisme ha esdevingut el mètode d'optimització més utilitzat dins del context del deep learning. Això es degut, tal i com mostren els autors al presentar l'algorisme, als beneficis que comporta utilitzar l'Adam en problemes d'optimització no convexa. El mètode és fàcil d'implementar, és computacionalment eficient, té pocs requisits de memòria, és invariant al reescalament diagonal dels gradients i és molt adequat per problemes de gran dimensió tant per dades com de paràmetres. A més, també és adequat per funcions no estacionàries i problemes amb molt soroll o amb gradients escassos. Pel que fa als hiperparàmetres tenen interpretacions intuïtives i, generalment, requereixen d'ajustament.

En les següents seccions veurem com es defineix el mètode d'optimització Adam, la tècnica de correcció del biaix d'inicialització i la relació d'aquest amb les extensions del SGD AdaGrad i RMSProp. I, a continuació, analitzarem la convergència teòrica de l'algorisme.

4.6.1 Algorisme

Com ja hem dit, una característica atractiva de l'Adam és la simplicitat per implementar l'algorisme. Tot i això, aquest requereix d'un seguit de passos per tal d'actualitzar els paràmetres, per aquest motiu veurem primer el pseudocodi de l'algorisme i, tot seguit, veurem en detall cada un dels passos.

Algorithm 1 Adam, l'algorisme presentat a l'article [21]. Una bona configuració predeterminada pels problemes d'aprenentatge automàtic és: $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$ i $\epsilon = 10^{-8}$.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
1: $m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
2: $v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector)
3: $t \leftarrow 0$ (Initialize timestep)
4: **while** θ_t not converged **do**
5: $t \leftarrow t + 1$
6: $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t stochastic objective at timestep t)
7: $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
8: $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second moment estimate)
9: $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
10: $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second moment estimate)
11: $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)
12: **end while**
13: **return** θ_t (Resulting parameters)

A l'algorisme 1, g_t^2 consisteix en elevar al quadrat cada element del vector g_t, les operacions amb vectors són element a element i β_1^t, β_2^t són β_1, β_2 a la potència t respectivament. Vegem doncs en detall cada un dels passos de l'algorisme.

En primer lloc, veiem que l'algorisme requereix d'uns hiperparàmetres α que correspon al learning rate o mida del pas en actualitzar els paràmetres i β_1 i β_2 que corresponen als coeficients d'actualització de les estimacions del primer i segon moment del gradient respectivament. També requereix d'una funció objectiu, $f(\theta)$, estocàstica parametritzada per $\theta \in \mathbb{R}^n$ i diferenciable respecte els seus paràmetres. Per últim, necessitem un vector de paràmetres inicial θ_0 per iniciar el bucle d'actualització dels paràmetres.

En segon lloc, les tres primeres línies inicialitzen les variables a 0, aquestes seran utilitzades en les iteracions de l'algorisme. El vector del primer moment del gradient, m_0, mantindrà una estimació de la mitjana del gradient. El vector del segon moment del gradient, v_0, mantindrà una estimació de la variància descentralitzada del gradient. La variable t indicarà la iteració.

A continuació, entrem en el bucle d'actualització dels paràmetres el qual es repetirà fins que els paràmetres θ convergeixin als valors que minimitzen la funció f. Vegem ara cada un dels passos realitzats per l'algorisme per tal de fer una actualització dels paràmetres.

Primer de tot augmentem el nombre de la iteració (línia 5) i calculem el gradient de la funció f respecte els paràmetres θ avaluada al pas t, és a dir amb els valors de θ actuals que corresponen als de la iteració anterior (línia 6).

Tot seguit, actualitzarem els valors corresponents a les estimacions del primer i segon moment del gradient. La línia 7 mostra l'actualització del vector m_t el qual manté una mitjana dels gradients anteriors que decreix exponencialment similar a l'algorisme Momentum. Aquest, com ja hem dit, correspon a l'estimació del primer moment del gradient, la mitjana. La línia 8 mostra l'actualització del vector v_t el qual manté una mitjana dels quadrats dels gradients anteriors que decreix exponencialment similar als algorismes

Adadelta i RMSProp. Aquest, com ja hem dit, correspon a l'estimació del segon moment del gradient, la variància.

Tanmateix, els vectors m_t i v_t són inicialitzats com a (vectors de) 0 que comporta que els valors d'aquests estiguin esbiaixades cap a 0 especialment durant els passos inicials i, sobretot, quan els coeficients β_1 i β_2 són propers a 1. Per aquest motiu, amb l'objectiu de corregir el biaix d'inicialització dels vectors m_t i v_t dividim aquests vectors entre $(1-\beta_1^t)$ (línia 9) i $(1-\beta_2^t)$ (línia 10) respectivament tal i com es mostra en el desenvolupament de la secció 3 de l'article de referència [21].

D'aquesta manera, al final de cada iteració, s'actualitzen els paràmetres θ seguint la regla d'actualització de la línia 11. Podem veure que apareix un nou hiperparàmetre, ϵ, el qual s'utilitza per evitar la divisió entre 0.

Observem que aquesta regla d'actualització és semblant als mètodes que utilitzen passos de mida adaptativa, en particular l'AdaGrad correspon a una versió de l'Adam prenent $\beta_1 = 0$, β_2 infinitesimalment proper a 1 i substituint α per $\alpha_t = \alpha \cdot t_{-1/2}$.

L'algorisme Adam també té una estreta relació amb l'algorisme RMSProp quan aquest incorpora el terme momentum. Tot i això, aquests segueixen mantenint certes diferències: l'RMSProp amb momentum actualitza els paràmetres utilitzant un terme momentum del gradient reescalat mentre que l'Adam actualitza els paràmetres mitjançant les estimacions del primer i segon moment del gradient. A més, l'RMSProp amb momentum no aplica cap correcció del biaix d'inicialització que, amb valors de β_2 propers a 1, pot comportar actualitzacions massa grans i sovint divergència.

4.6.2 Anàlisi de la convergència

En aquesta secció introduirem la demostració de la convergència de l'algorisme Adam tot i que no entrarem en el seu profund anàlisis. La complexitat d'aquest anàlisi es deixa per a futurs treballs ja que mereix una atenció més detallada que la que ha permès aquest treball. Tot i això, veurem les idees que es presenten a l'article [21] per donar una idea d'aquest anàlisi.

Considerem la seqüència de funcions de cost convexes $f_1(\theta), \ldots, f_T(\theta)$ obtingudes a cada iteració de l'algorisme. L'objectiu de l'article es donar una cota en funció de T de

$$R(T) = \sum_{k=1}^{T} [f_k(\theta_k) - f_k(\theta^*] \tag{4.6.1}$$

amb $\theta^* = \arg\min_{\theta} \sum_{k=1}^{T} f_k(\theta)$. D'aquesta manera, quan $T \to \infty$ tenim que els iterats de l'algorisme Adam convergeixen al mínim de la funció a minimitzar.

El teorema assumeix que la funció a minimitzar té gradients acotats i que la distància entre els paràmetres generats per l'Adam són acotats. Així, imposant condicions sobre els hiperparàmetres de l'Adam podem proporcionar una cota de $R(T)$. La demostració d'aquest teorema té certes similituds amb la demostració de la convergència del SGD però, degut a la complexa regla d'actualització de l'Adam, els procediments i la cota obtinguda per l'Adam són de major complexitat. Per veure el desenvolupament d'aquest anàlisi, consultar l'article [21] (secció *10. Appendix*).

4.7 Optimitzadors d'última generació

Finalment, veurem alguns dels algorismes d'optimització més recents dins l'àmbit del deep learning. Els algorismes que mostrarem a continuació pertanyen al mateix grup d'algorismes que l'Adam, mètodes que adapten la taxa d'aprenentatge per cada un dels paràmetres, i busquen millorar-ne el rendiment. Passem a veure quins són aquests algorismes.

En primer lloc, l'algorisme *AdaMax*[21] és una extensió de l'algorisme Adam que es va presentar en el mateix article que aquest. Com hem vist, la regla d'actualització de l'Adam actualitza cada un dels paràmetres escalant els seus gradients amb un factor inversament proporcional a una norma (escalada) L^2 de cada un dels seus gradients actuals i passats. Aquesta regla d'actualització basada en la norma L^2 es pot generalitzar a una regla d'actualització amb una norma L^p. L'AdaMax agafa el cas particular en que $p \to \infty$ generant un nou algorisme senzill i estable proporcionant una cota més simple de la magnitud de les actualitzacions dels paràmetres.

L'AdaMax és gairebé idèntic a l'Adam a excepció de l'estimació del segon moment del gradient, v_t. En aquest cas, substitueix v_t per:

$$u_t = \max\{\beta_2 \cdot u_{t-1}, |g_t|\} \tag{4.7.1}$$

amb $u_0 = 0$. A més, també suprimeix els passos de correcció del biaix d'inicialització.

En segon lloc, l'algorisme *AdaBelief*[44] un algorisme que també extén l'algorisme Adam amb tres objectius: convergència ràpida com els mètodes adaptatius, bona generalització com l'SGD i estabilitat de l'entrenament. L'experimentació realitzada mostra la superació d'altres mètodes amb una convergència ràpida i alta precisió en la classificació d'imatges i en la modelació del llenguatge. Concretament, l'AdaBelief aconsegueix una precisió comparable al SGD en el conjunt de dades ImageNet. A més, en l'entrenament d'una GAN amb el dataset Cifar10, l'algorisme demostra una estabilitat i millora la qualitat de les mostres generades en comparació amb l'Adam.

L'AdaBelief també és gairebé idèntic a l'Adam amb la mateixa excepció que l'AdaMax. En aquest cas, es substitueix l'estimació del segon moment del gradient v_t per:

$$s_t = \beta_2 \cdot s_{t-1} + (1 - \beta_2) \cdot (g_t - m_t)^2 \tag{4.7.2}$$

d'aquesta manera, aconsegueix realitzar passos més llargs quan l'observació g_t és proper a la predicció m_t i passos més curts en cas contrari.

En últim lloc, el *MADGRAD*[45], un algorisme molt recent que ha demostrat un rendiment excel·lent en problemes d'optimització del deep learning en múltiples camps: tasques de classificació, tasques imatge-a-imatge a visió per computador i models recurrents i emmascarats bidireccionalment en el processamen del llenguatge natural. Per cada una d'aquestes tasques, el MADGRAD iguala i, inclús, millora el rendiment del SGD i de l'Adam en les fases d'avaluació.

L'algorisme consisteix en una combinació de diferents tècniques que s'apliquen al mètode AdaGrad per abordar cada una de les seves mancances quan s'aplica a problemes d'optimització en el deep learning. Aquestes tècniques, així com la consetrucció del mètode, es troben explicades en detall a la secció 5 de l'article de referència, [45].

Capítol 5

Metodologia

En aquesta secció veurem detalladament els algorismes de Deep Reinforcement Learning (DRL), optimitzadors, configuracions i procediments que utilitzarem per realitzar els experiments i, d'aquesta manera, avaluar tant el rendiment dels algorismes de DRL com l'efecte del mètode d'optimització utilitzat. Els experiments es realitzaran per cada un dels algorismes de DRL que definirem, per cada un d'ells realitzarem diferents execucions utilitzant diferents mètodes d'optimització per actualitzar els paràmetres de les xarxes neuronals.

En primer lloc, veurem el conjunt de dades del qual disposem i com es defineixen els entorns amb els quals interactuarà l'agent determinat per a cada experiment. D'una banda, veurem quina informació contenen les dades proporcionades i com realitzarem la divisió d'aquestes en dos conjunts disjunts, entrenament i avaluació. D'altra banda, veurem com es defineixen els entorns així com les modificacions i adaptacions de l'entorn per potenciar l'aprenentatge dels agents.

En segon lloc, definirem, de manera teòrica, els algorismes de DRL que utilitzarem, el Joint PPO i el Rainbow. Tot seguit, per cada un d'ells, veurem com hem aplicat aquests algorismes a la pràctica mostrant les modificacions específiques per cada algorisme i definint-ne els seus hiperparàmetres.

En tercer lloc, enumerarem els mètodes d'optimització que hem vist anteriorment i utilitzarem pels experiments. Veurem els aspectes tècnics i hiperparàmetres usats a nivell pràctic.

Per últim, definirem un conjunt de mètriques per tal d'avaluar el rendiment, tant dels algorismes de DRL com dels mètodes d'optimització. Aquestes mètriques estaran definides d'una manera determinada i uniforme amb l'objectiu de poder comparar els experiments realitzats.

5.1 Dades

El conjunt de dades utilitzat per realitzar els experiments consistirà en un conjunt d'entorns definits pels jocs: *Sonic The Hedgehog™*, *Sonic The Hedgehog™2* i *Sonic 3 & Knuckles*. Observem que no es tracta d'un conjunt de dades a partir de les quals els algorismes de DRL aprendran certes característiques sinó que, simplement, indicaran quin és l'entorn amb el que interactuarà l'agent definit per l'algorisme de DRL.

A la secció 3.2 hem vist que la referència de cada entorn és una tripleta (ROM, zona, acte) a la qual anomenem nivell. Com hem vist disposem d'un conjunt de 58 nivells i, per tant, 58 entorns en els quals podem executar l'algorisme DRL corresponent.

Amb l'objectiu d'avaluar la capacitat de generalització i transferència de coneixements de l'agent dividirem el conjunt de nivells en un conjunt d'entrenament i en un d'avaluació. D'aquesta manera, en el procés d'aprenentatge usarem el primer conjunt i, seguidament, quan procedim a l'avaluació de l'agent usarem els nivells d'avaluació. Aquests conjunts s'han escollit aleatòriament tal i com es plantejava en el concurs agafant zones amb més d'un acte a l'atzar i, tot seguit, escollint un d'aquests actes a l'atzar. Així, ens quedarem amb 48 nivells d'entrenament i 10 nivells de avaluació, els quals podem veure a l'annex B.

5.2 Entorn

L'entorn amb el qual interactuarà l'agent ve definit per la llibreria de Python Gym-Retro[1]. Aquesta llibreria ens permet convertir videojocs clàssics en entorns Gym amb els quals desenvolupar algorismes de reinforcement learning. Els entorns Gym comparteixen una interfície genèrica definint una sèrie de funcions bàsiques per a la interacció agent-entorn: *reset()* per restablir l'entorn al seu estat inicial i *step()* per executar una acció des de l'estat actual.

D'aquesta manera, cada nivell ens indica quin entorn construir amb el Gym-Retro. Més detalladament, la tripleta que defineix cada nivell fa referència a un estat guardat que correspondrà a l'estat inicial de l'entorn. Així, construirem l'entorn Gym corresponent al joc i estat inicial indicat pel nivell concret.

Disposem doncs d'un entorn bàsic amb el qual podríem entrenar el nostre agent. Tot i això, tal com ve donat al codi base del concurs, realitzarem una sèrie de modificacions a aquest entorn amb l'objectiu d'adaptar-lo al joc del Sonic i obtenir millors resultats.

En primer lloc, discretitzarem el conjunt d'accions disponibles de l'entorn. Cada acció representa una combinació de botons de la consol que, en aquest cas, és la *Sega Genesis* la qual disposa dels següents botons: *B, A, MODE, START, UP, DOWN, LEFT, RIGHT, C, Y, X, Z*. Però pel Sonic hem considerat el següent conjunt d'accions: {{}, {*LEFT*}, {*RIGHT*}, {*LEFT, DOWN*}, {*RIGHT, DOWN*}, {*DOWN*}, {*DOWN, B*}, {*B*}}.

En segon lloc, transformarem la definició dels estats de l'entorn. A l'entorn Gym, cada estat consisteix en una imatge RGB de 24 bits, en el Sonic les imatges són de 320x224 píxels. Aplicarem una transformació a cada un d'aquests estats, passarem la imatge a escala de grisos i la reduirem a una imatge de 84x84 píxels. Això ens permetrà reduir el consum de memòria mantenint les característiques suficients de cada estat.

En tercer lloc, modificarem la funció *step()* aplicant *sticky frame skip*. Als entorns Gym, la funció *step()* fa un progrés en el joc d'$\frac{1}{60}$è de segon. Primer aplicarem un *frame skip*, el qual consisteix en aplicar una mateixa acció sobre un nombre determinat d'estats consecutius. En el nostre cas, utilitzarem un *frame skip* de 4, d'aquesta manera, cada pas representarà un progrés en el joc d'$\frac{1}{15}$è de segon. A més, com els entorns determinístics són susceptibles a solucions trivials guiades, aplicarem *sticky frame skip* la qual afegeix certa aleatorietat a les accions que prèn l'agent. Igual que el frame skip, cada acció s'aplica

[1] https://github.com/openai/retro

sobre un nombre determinat d'estats consecutius, la diferència és que, amb probabilitat 0.25, una acció s'extendrà un estat més. És a dir, com hem dit aplicarem una mateixa acció sobre 4 estats consecutius, la component sticky aplicarà, una de cada 4 vegades, l'acció anterior sobre el següent estat.

Per últim, cal distingir entre els dos algorismes de DRL que utilitzarem ja que, a diferència de les modificacions anteriors, les següents no seran comunes en cada un d'ells.

5.3 Mètodes DRL

En aquesta secció, presentarem els algorismes DRL que utilitzarem durant la fase d'experimentació. Per cada un d'ells, veurem, de manera teòrica, el funcionament d'aquests algorismes i, tot seguit, descriurem les configuracions usades per realitzar els experiments.

5.3.1 Joint PPO

L'algorisme *Joint PPO* és una variant distribuïda de l'algorisme *Proximal Policy Optimization*[15] (PPO). El Joint PPO aprèn una única política executant paral·lelament un nombre determinat d'actors. A cada un d'aquests se'ls assigna un entorn amb el que interactuar i amb la condició de que no tots els actors interactuen amb el mateix entorn, és a dir, juguen en diferents nivells.

L'algorisme PPO pertany a una nova família dels *policy gradient methods* (mètodes de gradient de polítiques). Aquest alterna entre obtenir dades a partir de la interacció amb l'entorn i l'optimització d'una funció objectiu aproximada mitjançant l'ascens del gradient estocàstic (*stochastic gradient ascent*). El PPO manté dos models com l'Actor-Critic i executa múltiples actors en paral·lel, N, sobre un mateix entorn, l'algorisme és el següent:

Algorithm 2 PPO, l'algorisme DRL presentat a l'article [15]

for *iteration* $= 1, 2, \ldots$ **do**
  for *actor* $= 1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{old}}$ in environment for T time steps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt. θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{old} \leftarrow \theta$
end for

A cada iteració, cada un dels actors realitza T passos a l'entorn obtenint informació sobre aquest i calculant els estimadors d'avantatge $\hat{A}_i$, funcions que depenen de la recompensa i de la funció de valor. A continuació es construeix la funció objectiu, o de pèrdua, L i s'optimitza amb un mètode d'ascens del gradient estocàstic.

Aquesta funció de pèrdua combina la funció de pèrdua de la política i l'error de la funció de valor. A més, afegeix un terme *entropy bonus* (bonificació d'entropia) per assegurar l'exploració. D'aquesta manera la funció de pèrdua es construeix de la següent manera:

$$L = \hat{\mathbb{E}}_t\big[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\big] \tag{5.3.1}$$

on c_1, c_2 són coeficients de valor i d'entropia respectivament, S denota l'entropy bonus. Les funcions L^{CLIP} i L^{VF} corresponen a les funcions de pèrdua de la política i la funció de

valor respectivament. La L^{CLIP} es tracta de la *Clipped Surrogate Objective function*[15] i la L^{VF} es tracta de l'error quadràtic de la funció de valor.

Pel que fa a la pràctica, he agafat el codi base de l'algorisme PPO del repositori de github OpenAI/Baselines[2] afegint un paràmetre per indicar l'optimitzador a utilitzar. Pels experiments, he escalat les recompenses de l'entorn amb un factor constant petit, 0.01 per tal de transformar els avantatges a un rang adequat per les xarxes neuronals. L'arquitectura de la CNN és la mateixa que la utilitzada a [15] per l'Atari.

L'algorisme utilitza uns hiperparàmetres, els quals defineixen la configuració utilitzada, per l'experimentació usarem els hiperparàmetres de la Taula 5.1.

Hiperparàmetre	Valor
Num. Actors	48
Horizon (T)	1024
Mini-batch size	6144
Num. Epochs	3
Cliping Parameter (ϵ)	0.1
Discount (γ)	0.99
GAE parameter (λ)	0.95
VF coefficient (c_1)	0.5
Entropy coefficient (c_2)	0.01

Taula 5.1: Configuració d'hiperparàmetres de l'algorisme Joint PPO usada per executar els experiments

Per tal d'aconseguir la millor combinació d'hiperparàmetres s'acostumen a utilitzar tècniques com el *grid search* o *random search* els quals consisteixen en provar diferents combinacions de paràmetres i avaluar quins donen els millors resultats. En el meu cas, he utilitzat la combinació d'hiperparàmetres donada a les bases del concurs exceptuant una modificació en l'hiperparàmetre *Horizon* el qual he hagut de reduir de 4096 a 1024 per reduir el consum de memòria.

A tot això, per aconseguir el comportament Joint del PPO, durant l'entrenament, utilitzarem un tipus d'entorn definit en el mateix repositori que ens permet executar diferents entorns en paral·lel. Aquest entorn executa un procés per cada entorn proporcionat obtenint experiències, és a dir transicions de l'agent a l'entorn, de cada un d'ells. Tot seguit, a la fase d'avaluació del gradient i actualització dels paràmetres l'algorisme utilitza les experiències obtingudes de cada un dels entorns definits.

Per últim, veurem quin serà el procediment per executar els experiments amb aquest algorisme en concret. Cada experiment constarà d'un agent definit per l'algorisme Joint PPO i un mètode d'optimització. Com hem vist, l'algorisme Joint PPO requereix d'un mètode d'optimització per actualitzar els paràmetres de la política, així doncs realitzarem un seguit experiments on cada un disposarà d'un mètode d'optimització diferent.

El procediment per realitzar aquests experiments estarà dividit en una primera fase d'entrenament i una segona fase d'avaluació. A cada una d'aquestes fases, per tal d'avaluar la capacitat de transferència del coneixement, utilitzarem dos conjunts de nivells disjunts, aquests són els dos conjunts de nivells definits anteriorment: els d'entrenament i els d'avaluació. Així doncs, les dues fases estaran definides de la següent manera:

[2]https://github.com/openai/baselines

Entrenament Executarem l'agent, tal i com l'hem definit en aquesta secció, utilitzant tots els entorns d'entrenament en paral·lel i limitarem aquesta execució amb un nombre màxim de passos, en concret es realitzaran un màxim d'un milió de passos per nivell.

Avaluació En aquesta fase, utilitzarem el model de l'agent après a la fase anterior per executar l'agent en cada un dels nivells d'avaluació de manera independent, és a dir el coneixement obtingut a cada nivell d'avaluació no serà compartit amb la resta. Aquesta execució també estarà limitada a un milió de passos durant els quals l'agent podrà adquirir coneixement de l'entorn i millorar el seu comportament.

5.3.2 Rainbow

L'algorisme *Rainbow*[16] és una extensió del DQN el qual ha rebut diferents millores independents durant els últims anys. El Rainbow combina sis d'aquestes millores combinant-les en un únic mètode. A la secció 2.4.2 hem vist com es construeix l'algorisme DQN així doncs ens queda veure les millores que s'incorporen a aquest les quals milloren l'algorisme en termes d'eficiència de dades com de rendiment final. Les millores que incorpora el Rainbow són les següents:

Double Q-Learning[46] Consisteix en utilitzar dues xarxes diferents, una xarxa de polítiques usada per escollir les accions i una xarxa objectiu per obtenir els Q-valors. Això ajuda a millorar l'estabilitat i evitar oscil·lacions. A més, també ajuda a afrontar la sobreestimació del biaix de selecció de les accions ja que la xarxa objectiu sempre escull el Q-valor màxim.

Prioritized Experience Replay[47] Com ja hem vist el DQN utilitza un Replay Buffer per guardar les transicions i en el pas d'aprenentatge l'algorisme escull uniformement un conjunt de transicions d'aquest. Com es suggereix a l'article hi ha transicions que poden tenir un major impacte en l'aprenentatge. Així es proposa millorar el rendiment prioritzant les transicions del Replay Buffer que tenen un major potencial d'aprenentatge amb l'ús d'una mètrica com el *TD-error* o una estocàstica proposades al mateix article.

Dueling Networks[48] Divideixen el còmput del Q-valor en el càlcul de la funció d'avantatge i el de la funció de valor. Aquesta arquitectura té una capa de xarxes compartida seguida de dos fluxos separats, un per estimar el valor de l'estat i l'altre per estimar l'avantatge de l'acció. Finalment, aquests dos es combinen per calcular el Q-valor permetent una millor generalització de les accions.

Multi-Step Learning[12] Consisteix en realitzar més d'un pas per calcular i actualitzar els Q-valors. Aquesta variant multi-step generalment condueix a un aprenentatge més ràpid.

Distributional RL[49] Fins ara ens hem centrat en aprendre una estimació de les expectatives del retorn, és a dir els Q-valors. Tot i això, l'article mostra que aprendre una distribució del valor en lloc d'un valor fix proporciona una millor estabilitat, sobretot quan s'utilitza una aproximació de la funció de valor. D'aquesta manera, aquí els Q-valors s'aproximen com una distribució de probabilitats factoritzada.

Noisy Networks[50] Aquesta variant modifica la constitució de les capes de les xarxes neuronals introduint una variable paramètrica de soroll, com el soroll gaussià, incrustada a les capes. Això indueix una estratègia d'exploració eficaç per sobre dels mètodes tradicionals, com l'epsilon greedy.

A la pràctica, he utilitzat com a codi base la implementació de l'algorisme Rainbow

proporcionat al repositori de github Unixpickle/AnyRL-py[3] afegint, igual que amb el Joint PPO, un paràmetre per indicar l'optimitzador a utilitzar. Pels experiments, he utilitzat un entorn modificat de manera que les recompenses que rep l'agent es basen en la diferència entre la recompensa actual i la màxima obtinguda en l'episodi en curs, en el cas que aquesta fos negativa es retorna zero. D'aquesta manera, l'agent és recompensat per avançar durant el nivell però no és castigat per anar enrere en el nivell. Pel que fa a l'arquitectura mantenim la presentada a [16].

Igual que amb l'algorisme anterior, el rainbow defineix un conjunt d'hiperparàmetres als quals hem donat els valors mostrats a la Taula 5.2.

Hiperparàmetre	Valor
Min history to start learning	20K
Noisy Nets σ_0	0.5
Target Network Period	1024
Buffer size	500K
Exploration ϵ	0.1
Prioritization type	proportional
Prioritization exponent ω	0.5
Prioritization importance sampling β	0.4
Multi-step returns n	4
Distributional atoms	51
Distributional min/max values	[-200,200]
Discount factor	0.99
Replay Period	every 1 agent step
Minibatch size	16

Taula 5.2: Configuració d'hiperparàmetres de l'algorisme Rainbow usada per executar els experiments

Igual que l'algorisme anterior, generalment els valors d'aquests hiperparàmetres s'obté mitjançant tècniques com el grid search. En el meu cas, he utilitzat els paràmetres del concursant que va quedar en segona posició en el concurs el qual va modificar els paràmetres *multi-step returns* de 3 a 4, *Target Network Period* de 8192 a 1024 i *Minibatch size* de 32 a 16 del codi base proporcionat.

Per últim, cal destacar una característica important del procediment que he seguit per executar els experiments amb el Rainbow. Igual que abans els experiments consten d'aquest algorisme de DRL executat múltiples cops cada un d'ells amb un mètode d'optimització diferent. La característica important és que l'algorisme Rainbow no requereix d'un entrenament previ per obtenir bons resultats en els nivells d'avaluació. El concursant que va quedar en segon lloc va mostrar que l'algorisme Rainbow sense entrenament previ donava millors resultats que realitzant un entrenament previ utilitzant la variant Joint del Rainbow. Per aquest motiu, el procediment dels experiments per aquest algorisme serà diferent a l'anterior ja que només constarà d'una fase d'avaluació amb les mateixes condicions que el Joint PPO. Aquestes condicions són: l'agent serà executat amb un nombre màxim de passos d'un milió i cada nivell serà avaluat de manera independent.

[3]https://github.com/unixpickle/anyrl-py

5.4 Mètodes d'optimització

Passem a veure els mètodes d'optimització que utilitzarem pels experiments. Com hem vist, per cada algorisme DRL executarem diferents experiments, cada un amb un mètode d'optimització diferent. El conjunt d'optimitzadors que utilitzarem són els que hem definit de manera teòrica a la secció 4.5. A efectes pràctics utilitzarem la implementació de cada un dels optimitzadors donada per la llibreria de Python *Tensorflow*[4]. Els algorismes d'optimització que utilitzarem són els següents: Momentum, Nesterov Accelerated Gradient (NAG), AdaGrad, Adadelta i Adam. Les taules 5.3 i 5.4 mostren les configuracions d'hiperparàmetres utilitzades en els experiments pel Joint PPO i pel Rainbow respectivament.

Optimitzador	Hiperparàmetre	Valor
Momentum	Learning Rate	2e-4
	Momentum	0.9
NAG	Learning Rate	2e-4
	Momentum	0.9
AdaGrad	Learning Rate	2e-4
	Initial Accumulator Value	0.1
Adadelta	Learning Rate	2e-4
	Rho	0.95
	Epsilon	1e-8
Adam	Learning Rate	2e-4
	Beta1	0.9
	Beta2	0.999
	Epsilon	1e-8

Taula 5.3: Configuració d'hiperparàmetres dels algorismes d'optimització pel Joint PPO

Optimitzador	Hiperparàmetre	Valor
Momentum	Learning Rate	1e-4
	Momentum	0.9
NAG	Learning Rate	1e-4
	Momentum	0.9
AdaGrad	Learning Rate	1e-4
	Initial Accumulator Value	0.1
Adadelta	Learning Rate	1e-4
	Rho	0.95
	Epsilon	1e-8
Adam	Learning Rate	1e-4
	Beta1	0.9
	Beta2	0.999
	Epsilon	1e-8

Taula 5.4: Configuració d'hiperparàmetres dels algorismes d'optimització pel Rainbow

[4]https://www.tensorflow.org/

5.5 Mètriques

Les mètriques són una part fonamental dels experiments les quals ens permetran avaluar cada un dels experiments, fer comparacions i extreure resultats. Així doncs, passem a veure quin serà el procés d'avaluació i, tot seguit, definirem les mètriques que utilitzarem durant aquest procés.

Com hem pogut veure els dos algorismes seguiran un procediment similar en l'execució dels experiments, en particular els dos comparteixen una fase d'avaluació sobre el conjunt de nivells destinat a aquesta fase. Així doncs, les mètriques que definirem a continuació estan destinades a ser utilitzades en aquesta fase, indiferentment de si hi ha una primera fase d'entrenament.

Donat que l'objectiu principal del treball és avaluar el rendiment dels algorismes de DRL en funció del mètode d'optimització usat, ens interessa definir unes mètriques que ens permetin avaluar aquest objectiu. Així doncs, els aspectes principals que voldrem avaluar són la puntuació, el temps d'execució i l'evolució de la funció de pèrdua. Definim doncs les mètriques que mesuraran aquests aspectes:

La puntuació Aquesta mètrica consistirà en una mitjana de les puntuacions per episodi de cada un dels nivells d'avaluació. Aquesta puntuació s'obté tal com hem explicat a la Secció 3.2, de 0 a 9000 punts en funció del desplaçament horitzontal més un bonus de màxim 1000 punts per completar el nivell. Així, per cada milió de passos d'un nivell d'avaluació, calcularem la mitjana de recompenses obtingudes dels episodis de cada nivell. Això ens donarà una mitjana de la puntuació obtinguda per nivell. A continuació, calcularem la mitjana de puntuació de les mitjanes obtingudes per cada un dels nivells. Així obtindrem una mètrica que ens permetrà avaluar i comparar el rendiment dels experiments. A més, també disposarem de les gràfiques d'aprenentatge que ens permetran comparar l'evolució de la recompensa mitja dels algorismes en funció del temps.

Temps d'execució Aquesta mètrica mesurarà tant el temps total d'execució per experiment com la mitjana de temps transcorregut en actualitzar els paràmetres. D'una banda, per cada nivell d'avaluació realitzarem la mitjana de temps total d'execució dels nivells. D'altra banda, per cada nivell farem la mitjana del temps transcorregut en actualitzar els paràmetres, tot seguit calcularem una nova mitjana de les mitjanes obtingudes per cada un dels nivells. Així doncs, els valors obtinguts amb aquesta mètrica ens permetran mesurar l'eficiència del mètode d'optimització per actualitzar els paràmetres de la xarxa neuronal.

Evolució de la funció de pèrdua Aquesta mètrica consistirà en una representació gràfica de la funció de pèrdua, de l'algorisme DRL corresponent vistos a la secció 5.3, en funció del temps. D'aquesta manera, representarem l'evolució de la funció de pèrdua la qual ens permetrà mesurar l'eficàcia del mètode d'optimització per actualitzar els paràmetres de les xarxes neuronals.

Observem que diferenciarem entre dos tipus d'experiments, els experiments realitzats amb l'algorisme Joint PPO i els experiments realitzats amb l'algorisme Rainbow. Els experiments realitzats amb cada un dels algorismes DRL es diferenciaran pel mètode d'optimització determinat. Així doncs, les mètriques definides anteriorment ens permetran comparar els resultats de cada un dels experiments realitzats amb el mateix algorisme DRL ja que tant el procés d'actualització dels paràmetres com les funcions de pèrdua de cada algorisme DRL són diferents. Tot i això, podrem usar la primera mètrica, la puntuació, per comparar el rendiment dels algorismes DRL.

Capítol 6

Resultats

En aquesta secció, farem un anàlisi i comparació dels resultats, donats per les mètriques definides a la fase anterior, obtinguts durant l'execució de cada un dels experiments.

En els experiments volem comparar el rendiment de cada un dels algorismes DRL en funció del mètode d'optimització utilitzat dins del problema DRL definit pel videojoc del Sonic. Com hem vist l'Adam es troba al cim de l'evolució dels algorismes d'optimització usats durant l'experimentació. Aquest fet, juntament a que l'Adam sol ser l'algorisme predominant en la majoria d'àrees del deep learning, ens porta a suposar que l'Adam serà l'algorisme que mostrarà una major generalització que comportarà els millors resultats en aquesta fase d'experimentació.

Abans de veure els resultats de l'experimentació, amb l'objectiu de donar cert sentit a les puntuacions obtingudes pels algorismes, veurem una referència humana de jugadors avaluats en el mateix problema. Aquesta referència, donada pel propi concurs, disposava de quatre subjectes de prova els quals van jugar en cada un dels nivells de prova durant una hora. Abans de veure els nivells de prova, cada subjecte disposava de dues hores per practicar sobre els nivells d'entrenament. La taula D.1 de l'Annex D mostra les puntuacions mitjanes dels humans en cada un dels nivells d'avaluació. En aquesta, podem veure que la mitjana d'avaluació és 7438.2 ± 624.2.

6.1 Experiments amb Joint PPO

Vegem doncs els resultats dels experiments realitzats amb l'algorisme DRL Joint PPO. Aquest ha sigut avaluat segons les mètriques mencionades a la secció anterior amb cada un dels algorismes d'optimització també especificats anteriorment. La Taula 6.1 mostra els resultats escalars de les mètriques, és a dir la puntuació mitjana i les mitjanes de temps per nivell i per actualització. La Figura 6.1 mostra les corbes d'aprenentatge per cada un dels experiments, és a dir les puntuacions obtingudes en funció del nombre de passos realitzat per l'algorisme, i la Figura 6.2 mostra l'evolució de la funció de pèrdua de l'algorisme en funció del nombre de passos.

Com ja hem dit, la Taula 6.1 mostra els resultats de puntuació, temps per nivell i temps per actualització de cada un dels experiments. Aquests resultats mostren, empíricament, que l'Adam supera en rendiment la resta d'optimitzadors, mesurant aquest rendiment amb la puntuació. Pel que fa a les mètriques temporals, observem que els optimitzadors Momentum i NAG requereixen de menys temps per realitzar l'optimització de la funció

de pèrdua i actualització dels paràmetres.

A la secció 4 hem fet un anàlisi de cada un d'aquests algorismes, això ens permet dividir els optimitzadors en dos grups: optimitzadors basats en momentum i optimitzadors basats en l'adaptació de la mida de pas. En el primer grup tenim el Momentum i el NAG mentre que en el segon grup tenim l'AdaGrad, l'Adadelta i l'Adam. Així doncs, podem justificar que els optimitzadors basats en adaptar les mides de pas requereixen de més temps perquè introdueixen aquests càlculs d'adaptació de la mida del pas per cada un dels paràmetres a cada iteració.

	Adam	Adadelta	AdaGrad	NAG	Momentum
Puntuació	**1827.61**	876.07	1312.26	1395.76	1374.25
Temps per Actualització	27.9171	27.9268	27.6601	27.6232	**27.6186**
Temps per nivell	1055.81	1050.69	1050.19	**1048.35**	1049.08

Taula 6.1: Resultats escalars dels experiments amb el Joint PPO

A la Figura 6.1 podem veure les puntuacions obtingudes al llarg de cada experiment. Cada una de les corbes representa la mitjana de les corbes d'aprenentatge de cada nivell d'avaluació. Observem doncs que, a partir dels 200 mil passos, l'experiment amb l'Adam sobrepassa a la resta augmentant la puntuació obtinguda mentre que la resta es manté entre 1250 i 1500 punts sense millorar substancialment la puntuació obtinguda. Aquest aplanament de la corba pot significar que l'agent tendeix a explotar accions vistes anteriorment disminuint la capacitat d'exploració i, així, l'agent no aconsegueix grans millores en termes de puntuació.

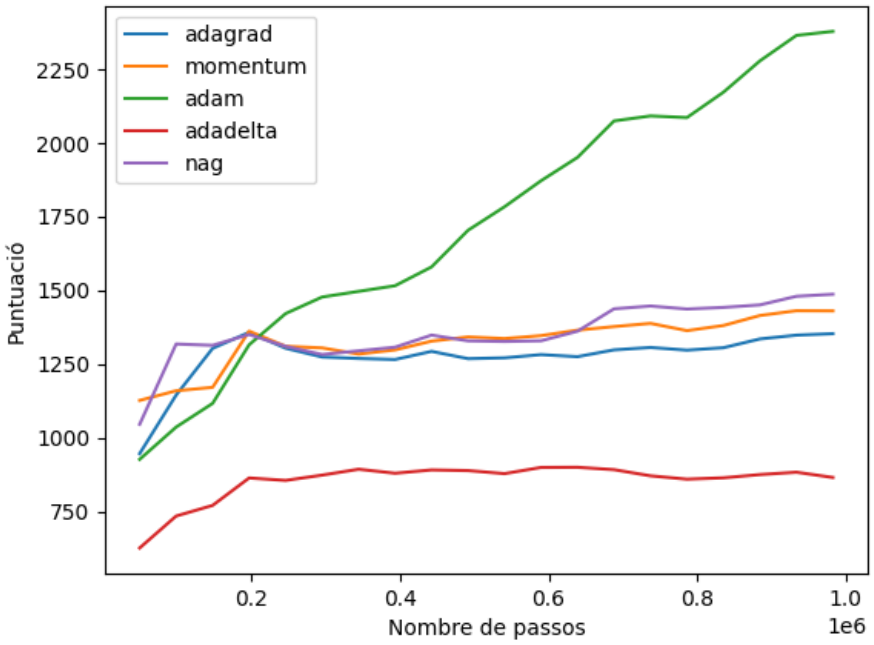

Figura 6.1: Corbes d'aprenentatge dels experiments amb el Joint PPO.

Tot i això, també cal considerar la Figura 6.2 la qual ens mostra, paral·lelament a

la corba d'aprenentatge, el valor de la funció de pèrdua a cada pas de temps. Aquesta ens permet fer noves suposicions respecte al comportament de les corbes d'aprenentatge. Observem que la corba de l'Adam té un valor inicial més alt reduint aquest valor quedant-se per sobre de la resta, les quals tenen una variació bastant reduïda. D'aquesta manera, podem intuir un possible *overfitting* (sobreajustament) sobre els nivells d'aprenentatge causant una limitació en la capacitat d'aprenentatge i generalització de l'agent.

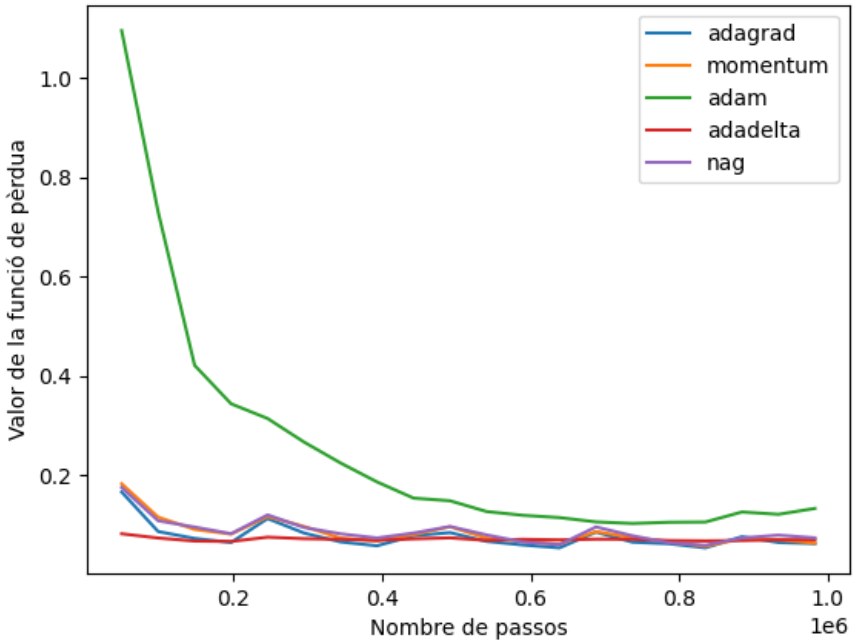

Figura 6.2: Evolució de les funcions de pèrdua dels experiments amb el Joint PPO.

Així doncs, d'aquests experiments podem extreure que, com esperàvem, l'ús de l'optimitzador Adam tot i no aconseguir els millors resultats en termes de velocitat o temps aconsegueix un millor rendiment en termes de puntuació i capacitat de generalització per part de l'agent. Per tant, hem demostrat, empíricament, que la millor l'Adam és el millor optimitzador pel Joint PPO en el benchmark del Sonic on hem realitzat els experiments.

6.2 Experiments amb Rainbow

Vegem ara els resultats dels experiments realitzats amb l'algorisme DRL Rainbow. Aquest, igual que el Joint PPO ha sigut avaluat segons les mètriques mencionades a la secció anterior amb cada un dels algorismes d'optimització també especificats anteriorment. La Taula 6.2 mostra els resultats escalars de les mètriques, és a dir la puntuació mitjana i les mitjanes de temps per nivell i per actualització. La Figura 6.3 mostra les corbes d'aprenentatge per cada un dels experiments, és a dir les puntuacions obtingudes en funció del nombre de passos realitzat per l'algorisme, i la Figura 6.4 mostra l'evolució de la funció de pèrdua de l'algorisme en funció del nombre de passos.

A la Taula 6.2 podem observar les puntuacions mitjanes, temps per nivell i temps per actualització assolits en cada un dels experiments. Igual que amb els experiments

anteriors, aquesta taula mostra, de manera empírica, que l'ús de l'Adam supera en rendiment la resta dels optimitzadors en termes de puntuació. Pel que fa a les mètriques temporals, podem observar que els temps per actualització amb el Rainbow són molt més reduïts que amb el Joint PPO però els temps per nivell són molt més elevats amb el Rainbow. Aquestes diferències són conseqüència de la naturalesa dels algorismes, la seva construcció i nombre d'actualitzacions realitzades. Tot i això, podem observar les mateixes característiques que amb els experiments realitzatas amb el Joint PPO, és a dir, els optimitzadors Momentum i NAG són els que requereixen menys temps i, en canvi, els algorismes que adapten les mides dels passos en les actualitzacions requereixen més temps per realitzar aquests càlculs.

	Adam	Adadelta	AdaGrad	NAG	Momentum
Puntuació	**3633.06**	768.21	1145.42	1552.15	2110.61
Temps per Actualització	0.0184	0.02	0.0177	**0.0176**	**0.0176**
Temps per nivell	21790.8	23176.84	20999.59	**20965.64**	20992.01

Taula 6.2: Resultats escalars dels experiments amb el Rainbow

La Figura 6.3 mostra les corbes d'aprenentatge dels experiments. De nou, podem veure que la corba d'aprenentatge del Rainbow quan utilitzem l'optimitzador Adam supera gairebé en tot moment la resta d'experiments. Tot i això, a diferència dels experiments amb el Joint PPO, les corbes de la resta d'experiments, a excepció de l'experiment amb l'Adadelta, no queden acotades per una puntuació que no poden superar. Així doncs, en aquest cas, tenim que els agents definits pel Rainbow mantenen certa capacitat d'exploració a la vegada que exploten les accions que ja sap que són bones, de nou l'Adadelta és l'excepció.

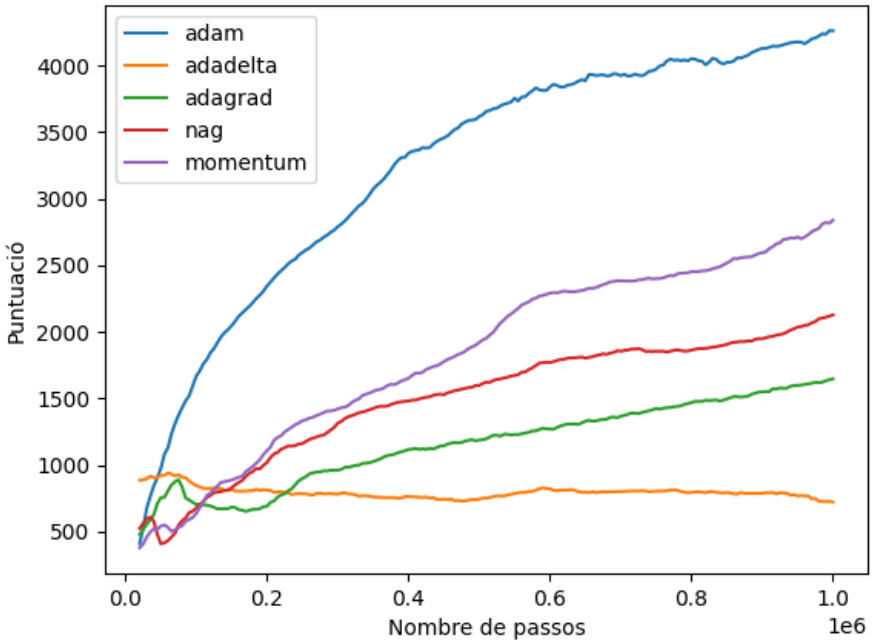

Figura 6.3: Corbes d'aprenentatge dels experiments amb el Rainbow.

La Figura 6.4 mostra els valors de la funció de pèrdua al llarg dels passos de l'experiment. Observem que les corbes tenen un comportament oscil·latori però es troba en un rang de valors petit així que no ho tindrem en compte a l'hora d'avaluar els resultats. També podem veure que la corba de l'Adam es troba per sota de la resta d'optimitzadors mentre que la resta es troben superposades entre elles dins d'un mateix rang. Això fa pensar que l'Adam actualitza els paràmetres de la xarxa neuronal de manera més precisa generant valors de la funció de pèrdua menors en les transicions posteriors.

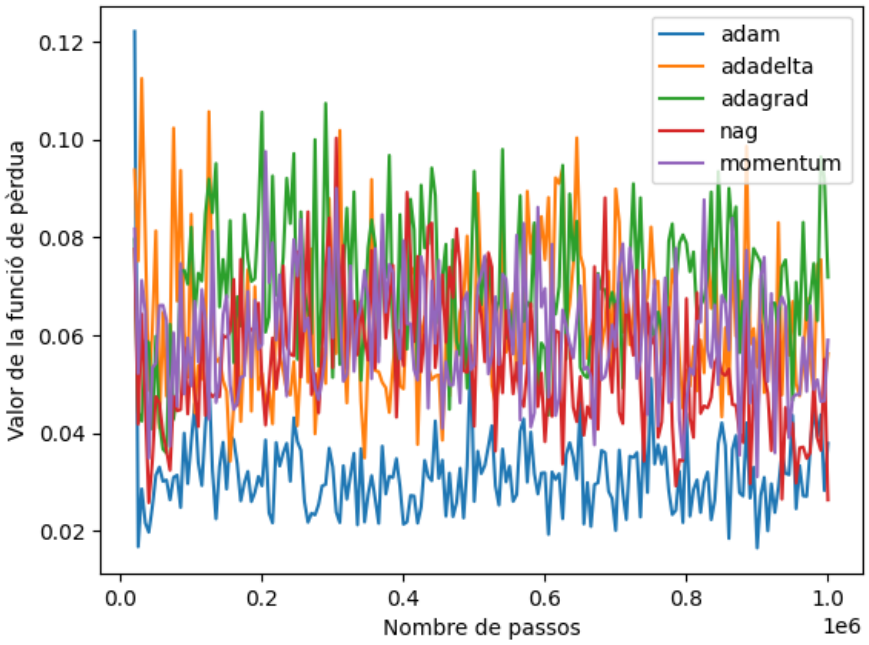

Figura 6.4: Evolució de les funcions de pèrdua dels experiments amb el Rainbow.

Igual que amb el Joint PPO, dels experiments realitzats amb el Rainbow podem extreure que l'Adam com a mètode d'optimització és la millor elecció. Hem pogut veure que tot i que pel que fa al temps no és el més ràpid però pel que fa a rendiment o puntuació, quan utilitzem l'Adam obtenim els millors resultats amb diferència convertint-lo en l'optimitzador ideal en aquest cas. I, d'aquesta manera, queda demostrat, de manera empírica, que l'Adam és el millor optimitzador pel Rainbow en el benchmark del Sonic on hem realitzat els experiments.

6.3 Joint PPO vs Rainbow

Finalment, vegem una comparativa dels resultats dels algorismes DRL. Com ja hem dit, per cada un dels algorismes DRL, obtenim els millors resultats quan utilitzem l'Adam com a algorisme d'optimització. Així doncs, vegem una comparativa dels resultats del Rainbow i Joint PPO dels experiments realitzats en aquest treball.

Ara podem veure a la Taula 6.3 les puntuacions, temps per nivell i temps per actualització pels experiments realitzats amb l'Adam del Joint PPO i del Rainbow. La Figura

6.5 mostra les corbes d'aprenentatge d'aquests experiments i la Figura 6.6 l'evolució de la funció de pèrdua.

Fent referència a la Taula 6.3 podem observar que la puntuació mitjana pel Rainbow és molt superior a la del Joint PPO, és gairebé el doble. Així doncs el Rainbow supera el Joint PPO en aquest aspecte en els experiments realitzats. Pel que fa a les mètriques temporals també podem observar grans diferències. En primer lloc, observem que els temps per actualitzar els paràmetres són molt diferents. Això es degut a la construcció dels algorismes dificultant la seva comparació. Respecte als temps per nivell podem observar que el Joint PPO és molt més ràpid que el Rainbow. En aquest cas, les mitjanes són comparables ja que aquestes representen la mitjana de temps en realitzar el milió de passos per nivell. Així doncs, aquesta taula ens mostra que el Rainbow aconsegueix puntuacions millors que el Joint PPO a canvi de temps de processament.

	Joint PPO	Rainbow
Puntuació	1827.61	**3633.06**
Temps per Actualització	27.9171	**0.0184**
Temps per nivell	**1055.81**	21790.8

Taula 6.3: Resultats escalars dels experiments amb l'Adam del Joint PPO i del Rainbow.

La Figura 6.5 ens mostra l'evolució de les puntuacions al llarg de l'avaluació de l'-experiment. Com ja hem dit, el Rainbow presenta puntuacions superiors al Joint PPO exceptuant l'inici ja que, inicialment, el Rainbow realitza únicament exploració.

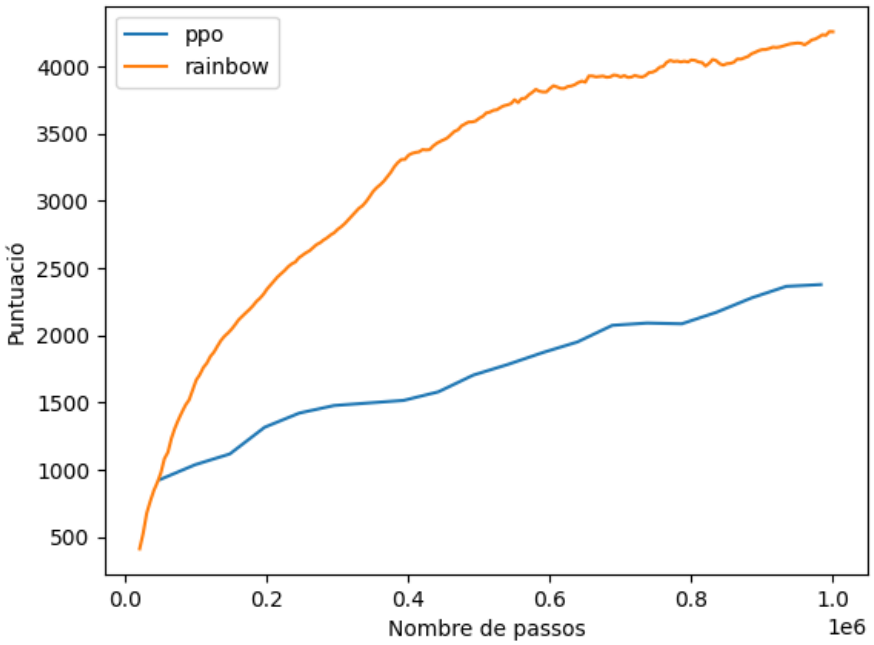

Figura 6.5: Corbes d'aprenentatge dels experiments amb l'Adam del Joint PPO i del Rainbow.

Tot seguit, la Figura 6.6 mostra l'evolució de la funció de pèrdua de cada un d'ells.

Observem que la corba del Rainbow es troba per sota de la del Joint PPO en tot moment. Aquesta comparació caldria analitzar-la amb detall ja que les funcions de pèrdua de cada un dels algorismes són diferents i caldria veure fins quin punt són comparables. Tot i això, com hem vist que el Rainbow obté millors puntuacions que el Joint PPO, podríem justificar que la corba del Rainbow es trobi per sota de la del Joint PPO per la relació entre la puntuació i les accions predites per l'algorisme. Realitzar una acció amb major puntuació generarà una pèrdua menor que una acció amb menor puntuació.

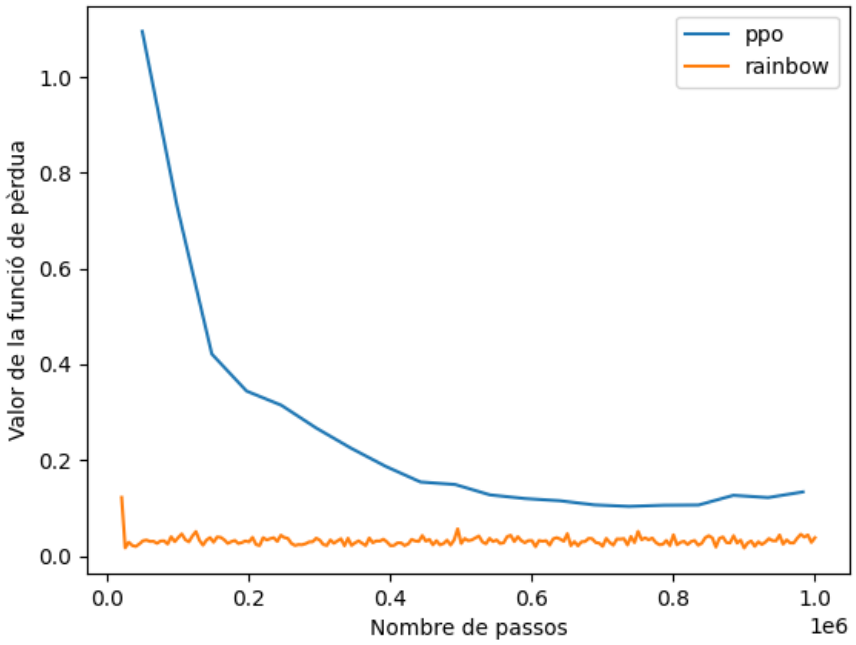

Figura 6.6: Evolució de les funcions de pèrdua dels experiments amb l'Adam del Joint PPO i del Rainbow.

Així doncs, podem concloure que el Rainbow obté millors resultats en termes de puntuació tot i que el Joint PPO és més ràpid. En el nostre cas, podem dir que el Rainbow és millor ja que no hem limitat el temps d'execució però podria donar-se el cas en què el temps fos limitat i, llavors el Joint PPO seria més adequat. Tot i això, cal tenir en compte que es tracta d'un cas concret amb una combinació d'hiperparàmetres determinada la qual no té perquè ser la més òptima. És més, prenent com a referència el concurs en què s'ha basat el treball, el Retro Contest, observem que les puntuacions obtingudes en aquest treball estan lluny dels guanyadors. Les solucions dels guanyadors, primer i segon lloc, consisteixen en una variant del Joint PPO i del Rainbow respectivament. La pàgina de resultats del concurs[1] mostra com el Joint PPO supera en rendiment el Rainbow mantenint la superioritat de velocitat del Joint PPO sobre el Rainbow.

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

Capítol 7

Conclusions

En aquest treball s'han exposat dos tipus de problemes: el problema de reinforcement learning en el videojoc del Sonic i el problema d'optimització del deep learning. L'objectiu principal del treball ha buscat analitzar l'efecte del mètode d'optimització en el rendiment dels algorismes DRL. Així doncs, el treball ha volgut fer un doble anàlisi. D'una banda, l'anàlisi d'un problema concret del DRL basat en el concurs Retro Contest. D'altra banda, l'estudi del problema d'optimització, una part fonamental del deep learning i, en conseqüència, de totes les aplicacions d'aquest. I, a diferència de l'objectiu principal del concurs, aquest treball ha buscat donar un estudi matemàtic formal d'aquest problema així com dels mètodes d'optimització per resoldre'l.

Inicialment, hem exposat la base teòrica del reinforcement learning: els Processos de Decisió de Markov i les solucions clàssiques per resoldre'ls. Seguidament, hem introduït el deep reinforcement learning i d'una sèrie d'algorismes els quals incorporen el deep learning a les solucions clàssiques. Tot això, ens ha permès formular el problema definit pel Sonic com un Procés de Decisió de Markov i idear el procediment per tal de resoldre'l.

Després, hem explorat la base matemàtica del problema d'optimització inherent al deep learning així com la solució bàsica, el Gradient Descent. Hem explorat tres de les variants d'aquest de les quals el Mini-Batch Gradient Descent n'és el més popular. Tot seguit hem fet una visió general de l'evolució dels algorismes veient-ne els més comuns en aquest tipus de problemes d'optimització: el Momentum, el Nesterov Accelerated Gradient, l'AdaGrad, l'Adadelta, l'RMSProp i l'Adam.

Respecte a l'algorisme Adam, hem analitzat el funcionament de l'algorisme veient que presenta una major complexitat que la resta d'optimitzadors vistos. Malauradament, l'anàlisi de la convergència d'aquest ha presentat un repte que no s'ha pogut assolir en aquest treball. Tot i això, hem introduït aquest anàlisi però, degut a la seva complexitat, no ha sigut possible entrar en el seu profund estudi. Així doncs, deixarem aquest estudi per a futurs treballs ja que mereix una atenció més detallada que la que ha permès aquest treball.

Seguidament, hem introduït una de les plataformes de recerca del reinforcement learning en videojocs, el Gym Retro. Aquesta ens ha permès generar els entorns de proves per a l'experimentació amb el videojoc del Sonic. L'ús d'aquesta plataforma, ha permès comprovar el rendiment d'algorismes DRL d'última generació com el Rainbow i el Joint PPO en el benchmark del Sonic.

Els experiments realitzats en aquest treball confirmen, de manera empírica, que tant

el Joint PPO com el Rainbow obtenen millors resultats quan l'algorisme d'optimització utilitzat és l'Adam. Hem pogut observar que, quan s'utilitza l'Adam com a optimitzador, l'agent definit pels algorismes DRL obté les millors puntuacions aprenent a generalitzar en diferents nivells del Sonic.

Com a resultat dels experiments també hem pogut comparar el rendiment del Joint PPO amb el del Rainbow usant l'Adam com a optimitzador. Aquests resultats mostren que el Rainbow supera el rendiment del Joint PPO en puntuació tot i ser més lent. Tot i això, els resultats del concurs mostren una variant del Joint PPO superant el rendiment del Rainbow. D'aquesta manera, aquest treball no permet concloure quin algorisme DRL és millor en el benchmark del Sonic. De tota manera, l'algorisme Rainbow no requereix d'una fase d'entrenament obtenint molts bons resultats. Així doncs, crec que el Rainbow seria una bona elecció per seguir investigant en aquest treball.

Recopilant, podem concloure com a resultat d'aquest treball que l'elecció del mètode d'optimització en cada problema és molt important per tal d'aconseguir els millors resultats. Els experiments mostren diferents rendiments dels algorismes DRL en funció del mètode d'optimització. Això limita la capacitat d'aprenentatge de l'agent i, en conseqüència, impedeix que l'agent millori els resultats en iteracions posteriors. Per tant, hem pogut veure de manera empírica que Adam és la millor elecció dels optimitzadors introduïts en aquest treball.

7.1 Treball Futur

Aquest treball ha servit per aprofundir en l'aplicació del DRL en els videojocs i el problema d'optimització del deep learning. Tot i això, aquest treball no ha pogut cobrir tots els aspectes a causa de la falta de temps. Seguidament enumero possibles continuacions o ampliacions que m'hauria agradat poder incloure en aquest treball:

- Analitzar detalladament la convergència de l'algorisme d'optimització Adam.
- Estudiar més exhaustivament el comportament dels optimitzadors a nivell pràctic. Monitoritzar, tant en l'entrenament com en l'avaluació, l'evolució de les funcions de pèrdua a cada actualització de l'algorisme DRL per comparar el comportament iteratiu de cada un dels optimitzadors.
- Ajustament d'hiperparàmetres per aconseguir millors resultats. Usar tècniques com el grid search o l'optimització bayesiana que construeix un model probabílistic de la funció que relaciona els valors dels hiperparàmetres amb l'objectiu avaluat en un conjunt d'avaluació.
- Comprovar si és possible transferir el coneixement amb jocs similars al Sonic, és a dir altres jocs de plataformes com per exemple el *Super Mario Bros.*
- Utilitzar altres algorismes DRL com el *Twin Delayed DDPG (TD3)*[51] un algorisme que ha donat bons resultats en altres tipus de tasques. Durant el desenvolupament del treball vaig provar d'utilitzar aquest algorisme pel Sonic però a causa de diversos problemes vaig haver d'excluir-lo.
- Executar nous experiments amb els algorismes d'optimització d'última generació mostrats a la secció d'optimització: l'AdaMax, l'AdaBelief i el MADGRAD.

Aquestes possibles continuacions inclouen anàlisis més profunds de determinades seccions, noves propostes per provar diferents mètodes o simplement curiositat per ampliar coneixement, les quals complementarien el treball realitzat.

Annexos

Annex A

Utilització del codi

En aquest annex proporcionaré els passos necessaris per preparar l'entorn de desenvolupament i proves del codi que complementa aquesta memòria. A més, també mostraré les diferents maneres d'executar aquest codi. Tot i això, el codi disposa d'un fitxer *README.md* on s'indiquen els mateixos passos i usos del codi.

A.1 Configuració de l'entorn

El codi ha estat desenvolupat en Python. Per aquest motiu, el codi té els següents requeriments:

- Python 3.7
- pip
- Virtualenv

Un cop instal·lats els requeriments enumerarem el seguit de passos per tal de tenir l'entorn configurat. Suposarem que el codi ja està descarregat i executarem els següents passos dins la carpeta d'aquest. Els passos seran els següents:

1. Crear un virtualenv amb la versió de Python 3.7 i activar-lo.
2. Instal·lar la dependència de Python *gym-retro*.
3. Descarregar i importar les ROMS del Sonic (aquestes es troben al directori ROMS del codi).
4. Instal·lar les dependències del retro contest.
5. Instal·lar tensorflow 1.14 (versió gpu si es disposa de gpu dedicada).
6. Instal·lar OpenCV.
7. Instal·lar dependències de Python *numpy, pandas i matplotlib*.

Les següents línies de codi mostren com realitzar els passos anteriors amb un terminal d'un sistema operatiu basat en Linux.

```
# Suposem que ens trobem a la carpeta base del codi
# Pas 1
virtualenv <env-name> -p <path-to-python3.7>
source <env-name>/bin/activate
# Pas 2
pip install gym-retro
# Pas 3
python -m retro.import ./ROMS
# Pas 4
git clone --recursive https://github.com/openai/retro-contest.git
pip install -e "retro-contest/support[docker,rest]"
# Pas 5
pip install tensorflow-gpu==1.14 # Si hi ha GPU dedicada disponible
pip install tensorflow==1.14 # En cas contrari
# Pas 6
pip install opencv-python
# Pas 7
pip install numpy
pip install pandas
pip install matplotlib
```

A.2 Executar el codi

Per executar els experiments disposem d'un script de Python *run.py* el qual ens permetrà executar tant la fase d'entrenament com la d'avaluació. Aquest accepta els següents paràmetres:

- *alg*: Indica l'algorisme DRL a utilitzar. Les opcions vàlides són: *ppo* o *rainbow*.
- *optim*: Indica l'algorisme d'optimització a utilitzar. Les opcions vàlides són: *adam*, *adadelta*, *adagrad*, *nag* i *momentum*.
- *save_path*: Indica el directori on guardar els models durant l'entrenament, per exemple el directori dins del projecte *RUNS*.
- *load_path*: Indica el directori des d'on carregar el model per a l'avaluació, és a dir el directori generat a l'entrenament *RUNS/ppo_adam_training_¡timestamp¿*
- *test*: Indica la fase: entrenament o avaluació. Aquest no rep cap paràmetre sinó que quan s'indica s'interpreta que es vol executar la fase d'avaluació i la d'entrenament en cas contrari.

Les següents comandes mostren com executar el codi.

```
# Mostrar informacio dels parametres
python run.py --help

# Executar entrenaments pel Joint PPO
python run.py --alg ppo --optim adam --save_path RUNS
# Executar avaluacions Joint PPO
```

```
python -W ignore run.py --alg ppo
    --load_path RUNS/ppo_training_1620984174 --test

# Executar entrenaments pel Rainbow
python run.py --alg rainbow --optim adam --save_path RUNS
# Executar avaluacions Rainbow
python -W ignore run.py --alg rainbow
    --load_path RUNS/rainbow_training_1620984174 --test
```

Per últim, per tal d'avaluar els experiments amb les mètriques definides disposem d'uns altres scripts Python que ens permeten construir les taules i imatges que apareixen en aquest treball. L'execució d'aquests scripts requereix d'un pas manual. El pas manual consisteix en crear una carpeta, per exemple *ppo_results* en aquesta carpeta hi copiarem les carpetes de cada un dels entrenaments dels quals volem computar les mètriques. Així doncs, si volem computar les mètriques dels experiments del Joint PPO realitzats amb l'Adam i el Momentum, copiaríem les carpetes *ppo_adam_training_¡timestamp¿* i *ppo_momentum_training_¡timestamp¿* dins la carpeta *ppo_results* i executaríem l'script *build_evaluation_metrics_ppo.py* de la següent manera:

```
python build_evaluation_metrics_ppo.py --dir ./ppo_results
```

Anàlogament, pels experiments realitzats amb el Rainbow també caldria fer aquest pas manual i, seguidament, executar l'script *build_evaluation_metrics_rainbow.py* de la següent manera:

```
python build_evaluation_metrics_rainbow.py --dir ./rainbow_results
```

Aquests scripts crearan al directori indicat una carpeta *results_¡timestamp¿* amb un fitxer que conté la taula (en latex) dels resultats escalars i dues imatges *png* que contenen les corbes d'aprenentatge i l'evolució de les funcions de pèrdua.

Annex B

Conjunt de nivells del Sonic

ROM	Zone	Act
SonicTheHedgehog-Genesis	GreenHillZone	2
SonicTheHedgehog-Genesis	StarLightZone	3
SonicTheHedgehog-Genesis	ScrapBrainZone	1
SonicTheHedgehog2-Genesis	MetropolisZone	3
SonicTheHedgehog2-Genesis	HillTopZone	2
SonicTheHedgehog2-Genesis	CasinoNightZone	2
SonicAndKnuckles3-Genesis	LavaReefZone	1
SonicAndKnuckles3-Genesis	FlyingBatteryZone	2
SonicAndKnuckles3-Genesis	HydrocityZone	1
SonicAndKnuckles3-Genesis	AngelIslandZone	2

Taula B.1: Nivells d'avaluació del Sonic pels experiments

ROM	Zone	Act	ROM	Zone	Act
SonicTheHedgehog-Genesis	SpringYardZone	1	SonicTheHedgehog2-Genesis	HillTopZone	1
SonicTheHedgehog-Genesis	SpringYardZone	3	SonicTheHedgehog2-Genesis	CasinoNightZone	1
SonicTheHedgehog-Genesis	SpringYardZone	2	SonicTheHedgehog2-Genesis	WingFortressZone	
SonicTheHedgehog-Genesis	GreenHillZone	3	SonicTheHedgehog2-Genesis	AquaticRuinZone	2
SonicTheHedgehog-Genesis	GreenHillZone	1	SonicTheHedgehog2-Genesis	AquaticRuinZone	1
SonicTheHedgehog-Genesis	StarLightZone	2	SonicAndKnuckles3-Genesis	LavaReefZone	2
SonicTheHedgehog-Genesis	StarLightZone	1	SonicAndKnuckles3-Genesis	CarnivalNightZone	2
SonicTheHedgehog-Genesis	MarbleZone	2	SonicAndKnuckles3-Genesis	CarnivalNightZone	1
SonicTheHedgehog-Genesis	MarbleZone	1	SonicAndKnuckles3-Genesis	MarbleGardenZone	1
SonicTheHedgehog-Genesis	MarbleZone	3	SonicAndKnuckles3-Genesis	MarbleGardenZone	2
SonicTheHedgehog-Genesis	ScrapBrainZone	2	SonicAndKnuckles3-Genesis	MushroomHillZone	2
SonicTheHedgehog-Genesis	LabyrinthZone	2	SonicAndKnuckles3-Genesis	MushroomHillZone	1
SonicTheHedgehog-Genesis	LabyrinthZone	1	SonicAndKnuckles3-Genesis	DeathEggZone	1
SonicTheHedgehog-Genesis	LabyrinthZone	3	SonicAndKnuckles3-Genesis	DeathEggZone	2
SonicTheHedgehog2-Genesis	EmeraldHillZone	1	SonicAndKnuckles3-Genesis	FlyingBatteryZone	1
SonicTheHedgehog2-Genesis	EmeraldHillZone	2	SonicAndKnuckles3-Genesis	SandopolisZone	1
SonicTheHedgehog2-Genesis	ChemicalPlantZone	2	SonicAndKnuckles3-Genesis	SandopolisZone	2
SonicTheHedgehog2-Genesis	ChemicalPlantZone	1	SonicAndKnuckles3-Genesis	HiddenPalaceZone	
SonicTheHedgehog2-Genesis	MetropolisZone	1	SonicAndKnuckles3-Genesis	HydrocityZone	2
SonicTheHedgehog2-Genesis	MetropolisZone	2	SonicAndKnuckles3-Genesis	IcecapZone	1
SonicTheHedgehog2-Genesis	OilOceanZone	1	SonicAndKnuckles3-Genesis	IcecapZone	2
SonicTheHedgehog2-Genesis	OilOceanZone	2	SonicAndKnuckles3-Genesis	AngelIslandZone	1
SonicTheHedgehog2-Genesis	MysticCaveZone	2	SonicAndKnuckles3-Genesis	LaunchBaseZone	2
SonicTheHedgehog2-Genesis	MysticCaveZone	1	SonicAndKnuckles3-Genesis	LaunchBaseZone	1

Taula B.2: Nivells d'entrenament del Sonic pels experiments

Annex C

Demostracions de Convergència

C.1 Convergència del Gradient Descent

Teorema. *Sigui $f : \mathbb{R}^d \to \mathbb{R}$ una funció convexa i diferenciable, tal que el seu gradient és Lipschitz, $L > 0$, i $x^* = \arg\min_x f(x)$. Aleshores, els iterats de l'algorisme Gradient Descent amb mida de pas $\alpha \leq 1/L$ satisfan el següent:*

$$f(x_k) - f(x^*) \leq \frac{||x_0 - x^*||_2^2}{2\alpha k} \tag{C.1.1}$$

Demostració. *Primer de tot, per convexitat de la funció f, tenim, per l'afirmació 1:*

$$f(x_i) \leq f(x^*) + \nabla f(x_i)^T (x_i - x^*) \tag{C.1.2}$$

A més, com el gradient de f és Lipschitz, per l'afirmació 2,

$$\begin{aligned} f(x_{i+1}) &\leq f(x_i) + \nabla f(x_i)^T (x_{i+1} - x_i) + \frac{L}{2}||x_{i+1} - x_i||_2^2 \\ &= f(x_i) - \alpha||\nabla f(x_i)||_2^2 + \frac{L\alpha^2}{2}||\nabla f(x_i)||_2^2 \\ &= f(x_i) - \alpha(1 - L\alpha/2)||\nabla f(x_i)||_2^2 \\ &\leq f(x_i) - \frac{\alpha}{2}||\nabla f(x_i)||_2^2 \end{aligned} \tag{C.1.3}$$

la última desigualtat surt de que $L\alpha \leq 1$. Combinant ara les dues equacions anteriors i el fet que $\nabla f(x_i) = (1/\alpha)(x_i - x_{i+1})$ tenim:

$$\begin{aligned} f(x_{i+1}) &\leq f(x^*) + \nabla f(x_i)^T (x_i - x^*) - \frac{\alpha}{2}||\nabla f(x_i)||_2^2 \\ &= f(x^*) + \frac{1}{\alpha}(x_i - x_{i+1})^T (x_i - x^*) - \frac{1}{2\alpha}||x_i - x_{i+1}||_2^2 \\ &= f(x^*) + \frac{1}{2\alpha}||x_i - x^*||_2^2 - \frac{1}{2\alpha}\big(||x_i - x^*||_2^2 - 2\alpha\nabla f(x_i)^T (x_i - x^*) + ||\alpha\nabla f(x_i)||_2^2\big) \\ &= f(x^*) + \frac{1}{2\alpha}||x_i - x^*||_2^2 - ||x_i - x^* - \alpha\nabla f(x_i)||_2^2 \\ &= f(x^*) + \frac{1}{2\alpha}\big(||x_i - x^*||_2^2 - ||x_{i+1} - x^*||_2^2\big) \end{aligned} \tag{C.1.4}$$

Ara, sumant l'equació anterior per $i = 0, \ldots, k-1$ obtenim

$$\sum_{i=0}^{k-1} \big(f(x_{i+1}) - f(x^*)\big) \leq \frac{1}{2\alpha}\big(||x_0 - x^*||_2^2 - ||x_k - x^*||_2^2\big) \leq \frac{||x_0 - x^*||_2^2}{2\alpha} \tag{C.1.5}$$

Finalment, per l'equació C.1.3, $f(x_0), \ldots, f(x_k)$ no és creixent. Per tant, $f(x_k) - f(x^*) \leq f(x_i) - f(x^*)$ per tot $i > k$. Així,

$$k\big(f(x_k) - f(x^*)\big) \leq \frac{||x_0 - x^*||_2^2}{2\alpha} \tag{C.1.6}$$

que és la desigualtat que volíem provar. □

C.2 Convergència del Stochastic Gradient Descent

Teorema. *Sigui $f : \mathbb{R}^d \to \mathbb{R}$ una funció convexa i diferenciable, tal que el seu gradient és Lipschitz, $L > 0$, i $x^* = \arg\min_x f(x)$. Considerem un cas del Stochastic Gradient Descent (SGD) amb estimadors v_k, el vector d'actualització tal que $\mathbb{E}[v_k] = \nabla f(x_k)$, té variància acotada: $\forall k \geq 0, Var(v_k) = \mathbb{E}[||v_k||_2^2] - \mathbb{E}[v_k]_2^2 \leq \sigma^2$. Aleshores, $\forall k \geq 1$, els iterats de l'algorisme SGD amb mida de pas $\alpha \leq 1/L$ satisfan el següent:*

$$\mathbb{E}[f(\bar{x}_k)] \leq f(x^*) + \frac{||x_0 - x^*||_2^2}{2\alpha k} + \alpha\sigma^2 \tag{C.2.1}$$

amb $\bar{x}_k = (1/k)(x_1 + \ldots + x_k)$.

Demostració. *L'argument és similar a la convergència del GD. Per l'afirmació 2,*

$$\begin{aligned} f(x_{i+1}) &\leq f(x_i) + \nabla f(x_i)^T(x_{i+1} - x_i) + \frac{L}{2}||x_{i+1} - x_i||_2^2 \\ &= f(x_i) - \alpha \nabla f(x_i)^T v_i + \frac{L\alpha^2}{2}||v_i||_2^2 \end{aligned} \tag{C.2.2}$$

Prenent esperances a ambdós costats respecte l'elecció del vector v_i, tenim:

$$\begin{aligned} \mathbb{E}[f(x_{i+1})] &\leq f(x_i) - \alpha||\nabla f(x_i)||_2^2 + \frac{L\alpha^2}{2}\big(||\nabla f(x_i)||_2^2 + Var(v_i)\big) \\ &\leq f(x_i) - \alpha(1 - L\alpha/2)||\nabla f(x_i)||_2^2 + \frac{L\alpha^2}{2}\sigma^2 \\ &\leq f(x_i) - \frac{\alpha}{2}||\nabla f(x_i)||_2^2 + \frac{\alpha}{2}\sigma^2 \end{aligned} \tag{C.2.3}$$

on la última desigualtat surt de que $L\alpha \leq 1$. Ara, combinant les dues equacions anteriors obtenim

$$\mathbb{E}[f(x_{i+1})] \leq f(x^*) + \nabla f(x_i)^T(x_i - x^*) - \frac{\alpha}{2}||\nabla f(x_i)||_2^2 + \frac{\alpha}{2}\sigma^2 \tag{C.2.4}$$

Ara substituïm v_i a l'equació usant $\mathbb{E}[v_i] = \nabla f(x_i)$ i $||\nabla f(x_i)||_2^2 = \mathbb{E}[||v_i||_2^2] - Var(v_i) \leq \mathbb{E}[||v_i||_2^2] - \sigma^2$, així:

$$\begin{aligned} \mathbb{E}[f(x_{i+1})] &\leq f(x^*) + \mathbb{E}[v_i]^T(x_i - x^*) - \frac{\alpha}{2}\mathbb{E}[||v_i||_2^2] + \alpha\sigma^2 \\ &= f(x^*) + \mathbb{E}\Big[v_i^T(x_i - x^*) - \frac{\alpha}{2}||v_i||_2^2\Big] + \alpha\sigma^2 \end{aligned} \tag{C.2.5}$$

Ara, repetim els càlculs fets a l'anàlisi del GD (Equació C.1.4) completant els quadrats dels termes del mig:

$$\begin{aligned}\mathbb{E}[f(x_{i+1})] &\leq f(x^*) + \mathbb{E}\Big[\frac{1}{2\alpha}(||x_i - x^*||_2^2 - ||x_i - x^* - \alpha v_i||_2^2)\Big] + \alpha\sigma^2 \\ &= f(x^*) + \mathbb{E}\Big[\frac{1}{2\alpha}(||x_i - x^*||_2^2 - ||x_{i+1} - x^*||_2^2)\Big] + \alpha\sigma^2\end{aligned} \tag{C.2.6}$$

Aquesta equació és anàlega a l'Equació C.1.4 afegint el terme $\alpha\sigma^2$ (i prenent esperances). Així doncs, sumant les equacions per $i = 0, \ldots, k-1$ obtenim:

$$\sum_{i=0}^{k-1}(\mathbb{E}[f(x_{i+1}] - f(x^*)) \leq \frac{1}{2\alpha}\big(||x_0 - x^*||_2^2 - \mathbb{E}[||x_k - x^*||_2^2]\big) + k\alpha\sigma^2 \leq \frac{||x_0 - x^*||_2^2}{2\alpha} + k\alpha\sigma^2. \tag{C.2.7}$$

Finalment, per l'afirmació 3,

$$kf(\bar{x}_k) = kf\Big(\frac{x_1 + \ldots + x_k}{k}\Big) \leq f(x_1) + \ldots + f(x_k). \tag{C.2.8}$$

Així,

$$\sum_{i=0}^{k-1}(\mathbb{E}[f(x_{i+1}] - f(x^*)) = \mathbb{E}[f(x_1) + \ldots f(x_k)] - kf(x^*) \geq k\mathbb{E}[f(\bar{x}_k)] - kf(x^*). \tag{C.2.9}$$

Combinant les dues equacions anteriors ens queda:

$$\mathbb{E}[f(\bar{x}_k)] \leq f(x^*) + \frac{||x_0 - x^*||_2^2}{2\alpha k} + \alpha\sigma^2 \tag{C.2.10}$$

que és l'expressió que enunciava el teorema. □

Annex D

Puntuacions en els nivells d'avaluació

Nivell	Puntuació
SonicTheHedgehog-Genesis GreenHillZone Act2	8166.1 ± 614.0
SonicTheHedgehog-Genesis StarLightZone Act3	8597.2 ± 729.5
SonicTheHedgehog-Genesis ScrapBrainZone Act1	6413.8 ± 922.2
SonicTheHedgehog2-Genesis MetropolisZone Act3	6004.8 ± 440.4
SonicTheHedgehog2-Genesis HillTopZone Act2	8600.9 ± 772.1
SonicTheHedgehog2-Genesis CasinoNightZone Act2	8662.3 ± 1402.6
SonicAndKnuckles3-Genesis LavaReefZone Act1	6705.6 ± 742.4
SonicAndKnuckles3-Genesis FlyingBatteryZone Act2	6021.6 ± 1006.7
SonicAndKnuckles3-Genesis HydrocityZone Act1	7146.0 ± 1555.1
SonicAndKnuckles3-Genesis AngelIslandZone Act2	8758.3 ± 477.9
Mitjana dels nivells d'avaluació	7438.2 ± 624.2

Taula D.1: Detall dels resultats d'avaluació pels humans

Bibliografia

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[2] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep Learning. *Nature*, 521:436–44, 05 2015.

[3] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An Introduction to Deep Reinforcement Learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354, 2018.

[4] Kun Shao, Zhentao Tang, Yuanheng Zhu, Nannan Li, and Dongbin Zhao. A Survey of Deep Reinforcement Learning in Video Games, 2019.

[5] Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement Learning in Robotics: A Survey. *Int. J. Rob. Res.*, 32(11):1238–1274, sep 2013.

[6] William Yang Wang, Jiwei Li, and Xiaodong He. Deep Reinforcement Learning for NLP. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, pages 19–21, Melbourne, Australia, jul 2018. Association for Computational Linguistics.

[7] Alexander Bernstein, Evgeny Burnaev, and O. Kachan. *Reinforcement Learning for Computer Vision and Robot Navigation*, pages 258–272. 07 2018.

[8] Chien Yi Huang. Financial Trading as a Game: A Deep Reinforcement Learning Approach, 2018.

[9] Zhengxing Huang, W.M.P. van der Aalst, Xudong Lu, and Huilong Duan. Reinforcement learning based resource allocation in business process management. *Data & Knowledge Engineering*, 70(1):127–145, 2011.

[10] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, 2017.

[11] Noam Brown and Tuomas Sandholm. Safe and Nested Subgame Solving for Imperfect-Information Games, 2017.

[12] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning, 2016.

[13] MichałKempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning, 2016.

[14] Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. Gotta Learn Fast: A New Benchmark for Generalization in RL, 2018.

[15] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, 2017.

[16] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining Improvements in Deep Reinforcement Learning, 2017.

[17] Ning Qian. On the Momentum Term in Gradient Descent Learning Algorithms. *Neural Netw.*, 12(1):145–151, jan 1999.

[18] Y. NESTEROV. A method for unconstrained convex minimization problem with the rate of convergence $o(1/k^2)$. *Doklady AN USSR*, 269:543–547, 1983.

[19] John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12:2121–2159, 07 2011.

[20] Matthew D. Zeiler. ADADELTA: An Adaptive Learning Rate Method, 2012.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, 2017.

[22] A. CAUCHY. Methode generale pour la resolution des systemes d'equations simultanees. *C.R. Acad. Sci. Paris*, 25:536–538, 1847.

[23] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 edition, 2010.

[24] Martin L. Puterman. Chapter 8 Markov decision processes. In *Stochastic Models*, volume 2 of *Handbooks in Operations Research and Management Science*, pages 331–434. Elsevier, 1990.

[25] Richard Bellman. *Dynamic Programming*. Dover Publications, 1957.

[26] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.

[27] Richard S. Sutton. Learning to Predict by the Methods of Temporal Differences. *Mach. Learn.*, 3(1):9–44, aug 1988.

[28] Ludwig Boltzmann. *Studien über das Gleichgewicht der lebendigen Kraft zwischen bewegten materiellen Punkten*, volume 1 of *Cambridge Library Collection - Physical Sciences*, page 49–96. Cambridge University Press, 2012.

[29] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, may 1992.

[30] G. A. Rummery and M. Niranjan. On-Line Q-Learning Using Connectionist Systems. Technical Report TR 166, Cambridge University Engineering Department, Cambridge, England, 1994.

[31] Ian H. Witten. An adaptive optimal controller for discrete-time Markov environments. *Information and Control*, 34(4):286–295, 1977.

[32] Nicholas Metropolis and S. Ulam. The Monte Carlo Method. *Journal of the American Statistical Association*, 44(247):335–341, 1949.

[33] Juan C. Meza. Steepest Descent. *WIREs Comput. Stat.*, 2(6):719–722, nov 2010.

[34] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, feb 2015.

[35] L. O. Chua and T. Roska. The CNN paradigm. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 40(3):147–156, 1993.

[36] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[37] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic Policy Gradient Algorithms. *31st International Conference on Machine Learning, ICML 2014*, 1, 06 2014.

[38] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016.

[39] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization Methods for Large-Scale Machine Learning, 2018.

[40] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[41] Xiaoyu Li and Francesco Orabona. On the Convergence of Stochastic Gradient Descent with Adaptive Stepsizes, 2019.

[42] Dong C. Liu and Jorge Nocedal. On the Limited Memory BFGS Method for Large Scale Optimization. *Math. Program.*, 45(1–3):503–528, aug 1989.

[43] T. Tieleman and G. Hinton. Lecture 6.5-RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[44] Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James S. Duncan. AdaBelief Optimizer: Adapting Stepsizes by the Belief in Observed Gradients, 2020.

[45] Aaron Defazio and Samy Jelassi. Adaptivity without Compromise: A Momentumized, Adaptive, Dual Averaged Gradient Method for Stochastic Optimization, 2021.

[46] Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning, 2015.

[47] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized Experience Replay, 2016.

[48] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling Network Architectures for Deep Reinforcement Learning, 2016.

[49] Marc G. Bellemare, Will Dabney, and Rémi Munos. A Distributional Perspective on Reinforcement Learning, 2017.

[50] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy Networks for Exploration, 2019.

[51] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods, 2018.

[52] Martijn Van Otterlo and Marco Wiering. *Markov Decision Processes: Concepts and Algorithms*. Springer, 2012.

[53] Yuxi Li. Deep Reinforcement Learning, 2018.