Aprenentatge profund per reforç aplicat al control automàtic de la locomoció de robots bípedes simplificats en entorns simulats

Estudiant:
Rafael Jesús Castaño Ribes
Màster en Enginyeria Informàtica
Intel·ligència Artificial

Consultor:
Samir Kanaan Izquierdo

Professor responsable de l'assignatura:
Carles Ventura Royo

Data de Lliurament:
5 de Gener de 2021

FITXA DEL TREBALL FINAL

Títol del treball:	*Aprenentatge profund per reforç aplicat al control automàtic de la locomoció de robots bípedes simplificats en entorns simulats*
Nom de l'autor:	*Rafael Jesús Castaño Ribes*
Nom del consultor/a:	*Samir Kanaan Izquierdo*
Nom del PRA:	*Carles Ventura Royo*
Data de lliurament (mm/aaaa):	*01/2021*
Titulació o programa:	*Màster en Enginyeria Informàtica*
Àrea del Treball Final:	*Intel·ligència Artificial*
Idioma del treball:	*Català*
Paraules clau	Deep Reinforcement Learning Bipedal Locomotion Simulated Robotics

Resum del Treball (màxim 250 paraules): *Amb la finalitat, context d'aplicació, metodologia, resultats i conclusions del treball*

Context:

L'objecte d'estudi d'aquest Treball Final de Màster és el control autònom de la locomoció bípeda (*biped locomotion*) mitjançant la intel·ligència artificial.

Finalitat:

Conèixer l'estat de l'art d'aquest camp d'estudi i implementar una solució moderna a aquest problema en un entorn simplificat i simulat per programari.

Metodologia:

1) Es descriu amb detall el problema i es relaciona amb la disciplina de l'*Aprenentatge (Profund) per Reforç ((Deep) Reinforcement Learning)*.
2) S'analitzen els fonaments teòrics d'aquesta disciplina i els principals mètodes d'aplicació al problema en qüestió.
3) S'analitza la plataforma *OpenAI Gym* i els entorns que ofereix; s'accepta com a plataforma sobre la qual desenvolupar el producte.
4) S'analitzen diferents biblioteques disponibles per a DRL i se n'escull una (la *TF-Agents*).
5) S'escull l'algorisme a implementar (l'algorisme NAF), i es dissenya el producte a desenvolupar. El disseny inclou un conjunt d'eines necessàries per al seu funcionament. Així mateix, es desenvolupa un *wrapper* per

compactar sèries d'observacions de l'entorn, a l'estil de Mnih et al. (2015) amb les DQN contra Atari.

6) S'implementa el producte en Python.

Resultats:

1) El *wrapper* implementat té un efecte positiu sobre l'aprenentatge dels agents.
2) L'agent desenvolupat funciona correctament i és capaç de resoldre el problema quan es combina amb el *wrapper* implementat.

Conclusions:

- El DRL és una disciplina complexa, especialment quan l'espai d'accions del problema és continu.
- S'ha pogut aproximar una solució al problema mitjançant DRL.
- La biblioteca TF-Agents, malgrat en desenvolupament, ha estat molt útil per aprofundir en el coneixement del DRL i llurs components.

Abstract (in English, 250 words or less):

Context:
The subject of this Master's Thesis is the autonomous control of biped locomotion by means of artificial intelligence.
Purpose:
To know the *state of the art* of this field and to implement a modern solution to this problem in a simplified software-simulated environment.
Methodology:
1) The problem is described in detail and related to the *(Deep) Reinforcement Learning (DRL)* field.
2) Theoretical foundations of DRL and its main methods applicable are analyzed.
3) The *OpenAI Gym* platform and its environments are analyzed; They are accepted as the platform on which to develop the product.
4) Different libraries available for DRL are analyzed and one is chosen (the *TF-Agents* library).
5) The algorithm to be implemented is chosen (the NAF algorithm), and the product to be developed is designed. The design includes a set of tools needed for its operation. A wrapper is also developed to compact series of observations of the environment, based on Mnih et al. (2015) in their DQN vs Atari experiment.
6) The designed product is implemented in Python.

Results:
1) The implemented *wrapper* has a positive effect on the learning of the agents.
2) The developed agent works properly and is able to solve the problem when combined with the *wrapper*.

Conclusions:
• DRL is a complex discipline, especially when the action space of the problem is continuous.
• The problem can be approximatedly solved using DRL.
• The TF-Agents library, although under development, has been very useful in deepening in the knowledge of DRL and its components.

Índex

Llista de figures

1 Introducció

1.1 Context i justificació del Treball

Context

L'aplicació de la intel·ligència artificial a la robòtica és considerada per molts com un dels camps més significatius i excitants en el desenvolupament de la robòtica. Mentre que la intel·ligència artificial podria acabar generant algun dia ordinadors capaços de "pensar" com els humans, la seva aplicació a la robòtica podria acabar generant robots capaços d'actuar i dur a terme tot tipus de tasques d'una manera "similar" als humans (Bogue, 2014).

Dins el camp de la robòtica, els robots hominoides tenen el potencial de dur a terme tasques complexes en diferents tipus d'entorns degut a l'elevat nombre de graus de llibertat que els ofereix la seva estructura (Xi & Chen, 2020).

Les aplicacions d'aquests tipus de robots poden ser molt variades: des del treball en situacions hostils i de perill (com poden ser els entorns contaminats, la desactivació d'explosius, les missions de rescat i de salvament o, fins i tot, l'exploració espacial), a l'atenció a persones en situació de dependència (Bogue, 2014) (Silva, Santos, Matos, & Costa, 2014).

Un dels trets que permet a aquests tipus de robots de tenir un avantatge potencial respecte a altres tipus de robots és el bipedisme (en contraposició als robots basats en rodes, o els reptants), en oferir una via més flexible per lidiar amb obstacles complexes i terrenys irregulars durant el seu desplaçament (Silva, Santos, Matos, & Costa, 2014).

Per aconseguir aquest avantatge, aquests robots han de ser capaços d'adaptar-se a les diferents situacions que planteja cada tipus de terreny i mantenir la seva funcionalitat malgrat la presència d'obstacles, irregularitats i pertorbacions en l'entorn. En aquest sentit, diferents grups de recerca han treballat en el desenvolupament de controladors de la locomoció bípeda amb l'objectiu que siguin robustos, que proporcionin una major estabilitat i que permetin als robots adaptació a l'entorn de forma automàtica durant l'execució de les seves tasques (Xi & Chen, 2020).

Necessitat a cobrir

L'objecte d'estudi d'aquest Treball Final de Màster és el control autònom de la locomoció bípeda (*biped locomotion*[1]) mitjançant la intel·ligència artificial.

En concret, es pretén conèixer, a grans trets, quin és l'estat de l'art d'aquest camp de recerca i implementar una solució moderna a aquest problema en un entorn simplificat i simulat per programari en acordança a les metodologies actuals detectades.

[1] En la bibliografia, a locomoció bípeda també se la sol anomenar amb el mot "caminar" (*walking*)

Com res resol el problema de moment?

La locomoció bípeda és un problema complex en què hi intervenen molts elements, com són la generació del moviment a efectuar d'acord a l'objectiu del robot, el control i coordinació dels diferents efectors que permeten generar aquest moviment (origen de l'elevat nombre de graus de llibertat del problema), la percepció i interpretació de l'entorn, la planificació de la tasca, el manteniment de l'equilibri i la minimització de l'efecte de les pertorbacions, entre altres (Silva, Santos, Matos, & Costa, 2014).

Aquest problema ha estat adreçat al llarg del temps des de múltiples perspectives que seran analitzades en el capítol de l'estat de l'art d'aquesta memòria.

Aquestes metodologies es poden classificar de múltiples maneres, en funció d'on posen el focus. A tall d'exemple:

- En funció de l'existència o no d'un conjunt de dades previ del qual aprendre
  - Aprenentatge per demostració
  - Aprenentatge per assaig i error
  - Mètodes híbrids que combinen tots dos enfocaments
- En funció de si el mètode utilitza i/o genera un model per a l'aprenentatge
  - Mètodes basats en models (*model-based*)
  - Mètodes sense models (*model-free*)
  - Mètodes híbrids que combinen tots dos enfocaments
- En funció de l'algorisme d'aprenentatge
  - Algorismes genètics (*Genetic Algorithms*)
  - Aprenentatge per reforç (*Reinforcement Learning o RL*)
  - Aprenentatge profund per reforç (*Deep Reinforcement Learning o DRL*)

D'entre tots aquests paradigmes, en els darrers anys ha guanyat importància l'aplicació dels mètodes de DRL (García & Shafie, 2020). Aquest paradigma permet generar agents intel·ligents capaços d'adaptar-se a l'entorn a temps real amb un temps de resposta relativament baix, degut a què l'aprenentatge per dur a terme aquesta adaptació i de les polítiques a executar es produeix *off-line* (abans de posar el robot en acció en l'entorn real), de manera similar a com aprenen les xarxes neuronals convencionals (Nian, Liu, & Huang, 2020).

Així mateix, darrerament, alguns autors han incorporat un altre objectiu en aquest camp, que és el minimitzar les caigudes durant l'aprenentatge per tal de disminuir els danys soferts pel robot, com és el cas del paradigma del *Safe Reinforcement Learning* (García & Shafie, 2020).

Quin resultat es vol obtenir?

L'objectiu inicial d'aquest treball final de màster ha estat identificar i analitzar algunes de les metodologies més actuals en l'àmbit del control automàtic de la locomoció dels robots bípedes i aplicar aquest coneixement per generar un agent per a un model de robot bípede simplificat en un entorn físic simulat, fent especial

èmfasi en la integració d'algun dels paradigmes del DRL tenint en compte, a la vegada, la seguretat física del robot al llarg de l'aprenentatge.

Donades les limitacions de temps i de recursos, el model ha de ser simplificat i l'entorn ha de ser simulat. A més a més, els algorismes hauran de ser prou eficients com per funcionar en un maquinari d'àmbit domèstic. Per tots aquests motius, així com per simplificar la tasca de posada en marxa de l'entorn i que els resultats siguin comparables a altres projectes, es proposa utilitzar els models d'hominoides ja programats de la plataforma *OpenAI Gym*[2].

En concret, i a proposta dels consultors del treball, es proposa començar per l'entorn 2D anomenat BipedalWalker (Figura 1), seguit de l'entorn BipedalWalkerHardcore (Figura 2) i, en el cas d'obtenir resultats satisfactoris en un temps raonable, s'intentarà aplicar la solució a un entorn 3D, com el *Gym* proporcionat per l'entorn físic *pyBullet*[3], que inclou els entorns Walker2DBulletEnv (Figura 3, equivalent a la solució 2D *BipedalWalker* d'*OpenAI Gym*), i HumanoidBulletEnv (Figura 4).

Figura 1. Entorn BipedalWalker
(imatge capturada de https://gym.openai.com/envs/)

Figura 2. Entorn BipedalWalkerHardcore
(imatge capturada de https://gym.openai.com/envs/)

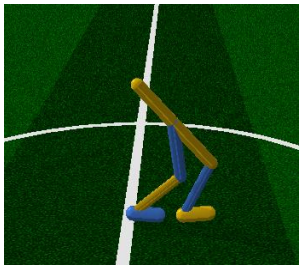


Figura 3. Entorn Walker2DBulletEnv
Imatge extreta de (Coumans & Bai, 2020)

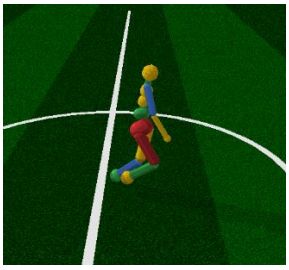


Figura 4. Entorn HumanoidBulletEnv
Imatge extreta de (Coumans & Bai, 2020)

Perquè és un tema rellevant?

La hiperautomatització i les "coses autònomes" (*autonomous things*) són dues de les 10 principals tendències tecnològiques estratègiques identificades per la consultora Gartner en el seu informe per al 2020 (Cearley, et al., 2019).

En aquest sentit, el desenvolupament de mecanismes eficaços i eficients per al control autònom de la locomoció dels robots bípedes i hominoides pot esdevenir

[2] https://gym.openai.com/
[3] https://pybullet.org/

un avenç important. Aquest control els permetria obtenir avantatges, per exemple, en entorns on es requereixi una resposta pràcticament reflexa o immediata (com pot ser l'adaptació ràpida a noves situacions o evitar accidents imminents del robot), quan hi pugui haver un retard important entre el robot i el controlador humà a càrrec (com pot ser en zones de difícil cobertura o en l'exploració d'altres planetes), o simplement en situacions en què es desitja que el robot sigui autònom i actuï per sí mateix, sense que l'humà a càrrec s'hagi de preocupar per definir les ordres relatives a aquest moviment.

Una altra tendència tecnologia estratègica per al 2020 inclosa en l'informe de la consultora Garnter és l'augmentació humana (*human augmentation*) (Cearley, et al., 2019). Els coneixements i tecnologies aplicats al control autònom de la locomoció bípeda també poden ser d'aplicació en el disseny de pròtesis intel·ligents (Bogue R., 2014), que encaixarien en la definició d'augmentació humana.

D'altra banda, tant en la disciplina de la Intel·ligència Artificial com en la de la Robòtica, existeix una gran fragmentació entre els seus diferents camps d'estudi, amb un elevat grau d'especialització dels investigadors en aspectes i problemes concrets dins de cada disciplina. La integració de diferents metodologies i paradigmes de cadascun dels subcamps d'una forma holística és un dels reptes més importants per a l'obtenció de robots autònoms i intel·ligents i esdevé en sí mateixa un camp de recerca rellevant (Bogue, 2014). En aquest sentit, el recull i anàlisi de metodologies punteres que afecten al problema des de diferents punts de vista, com és el cas de l'eficiència en la resolució del problema, d'una banda, i la seguretat física del robot, d'una altra, pretén respondre a aquesta disciplina emergent.

Finalment, el fet que la solució s'hagi de poder executar en un temps raonable en un ambient de treball domèstic, planteja un repte interessant per al problema alhora que apropar aquesta disciplina al concepte de la *citizen science*[4], fet que pot ser rellevant a nivell social.

[4] Per a més informació sobre el concepte de *citizen science*, veure (Gura, 2013)

1.2 Objectius del Treball

Objectiu general

Obtenir un agent autònom de control de la locomoció per a un model de robot bípede simplificat en un entorn simulat tot aplicant diferents enfocaments actuals de la Intel·ligència Artificial respecte a aquest problema.

Objectius específics

1. Conèixer els principals enfocaments actuals des de la Intel·ligència Artifical (IA) per resoldre el problema del control autònom de la locomoció bípeda en els robots hominoides i en els seus models simulats.

2. Conèixer els paradigmes del *Reinforcement Learning (RL)* i del *Deep RL (DRL)*, els problemes als que s'adreça i les tècniques actuals dins d'aquest camp, tot relacionant-ho amb el problema a resoldre.

3. Valorar les possibilitats de la plataforma *OpenAI Gym* i dels seus entorns bidimensionals *BipedalWalker* i *BipedalWalkerHardcore* en relació al problema escollit.

4. Identificar els requisits que ha de complir la solució proposada, tot tenint presents les limitacions de l'entorn on s'executarà aquesta solució (un entorn domèstic), el temps disponible, la plataforma escollida (*OpenAI Gym – BipedalWalkers*) i la possibilitat d'escalar la solució posteriorment a un model 3D (per exemple *pyBullet*).

5. Aplicar el coneixement adquirit per desenvolupar una solució al problema del control autònom de la locomoció bípeda en entorns simplificats 2D, tot integrant de forma holística els enfocaments escollits, tot valorant la possibilitat que un d'aquests enfocaments sigui el paradigma del Safe RL.

6. Executar el model i analitzar els resultats tant per cada enfocament de forma aïllada com per l'aplicació combinada d'aquests.

7. Escalar i aplicar la solució plantejada a un model de walker en entorn 3D , i comparar i analitzar els resultats respecte als obtinguts en els entorns 2D.

1.3 Enfocament i mètode seguit

El punt de partida de l'estudiant respecte al problema és el coneixement adquirit en el màster a partir de la matèria d'Intel·ligència Artificial Avançada. En els materials docents d'aquesta matèria es feia referència a l'aprenentatge per reforç i a la plataforma *OpenAI Gym*, però no es profunditzava en el seu estudi.

D'aquesta manera, esdevé necessari, per tant, iniciar el treball mitjançant un estudi de l'estat de l'art respecte a les solucions al problema plantejat.

Un cop coneguts els principals enfocaments actuals, caldrà analitzar el funcionament de la plataforma *OpenAI Gym* i valorar la seva idoneïtat de cara a desenvolupar una possible solució en el marc d'aquest TFM.

De la mateixa manera, caldrà analitzar també les diferents biblioteques disponibles per al desenvolupament de solucions de (*Deep*) *Reinforcement Learning* i decidir quina és la més idònia per al desenvolupament de la solució d'aquest TFM.

Amb la informació obtinguda i les decisions preses, es procedirà a dissenyar una possible solució que consisteix a implantar un agent intel·ligent mitjançant la biblioteca seleccionada en la plataforma OpenAI Gym, prèvia acceptació de la seva idoneïtat.

Un cop desenvolupada la solució, caldrà posar-la en pràctica mitjançant un conjunt d'experiments per tal de provar-ne el seu funcionament i eficiència, així com per trobar els valors dels paràmetres que permeten optimitzar-ne el resultat.

1.4 Planificació del Treball

D'acord als materials de referència per al TFM (Clarisó Viladrosa, 2013), la planificació d'aquest treball s'ha dut a terme seguint la metodologia de gestió de projectes de l'estàndard PMBOK (PMI, 2017) (Figura 5).

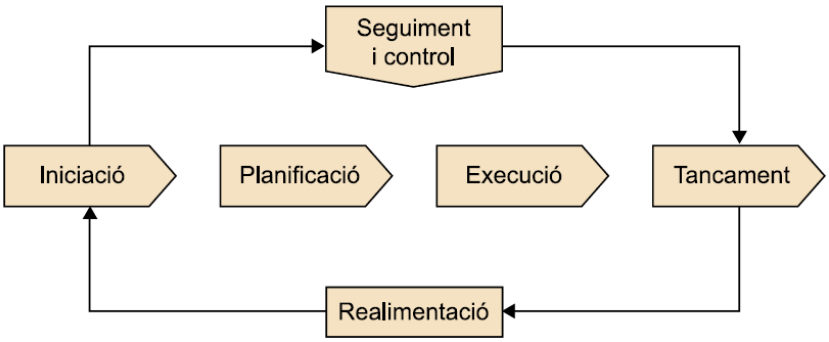


Figura 5. Cicle de vida de gestió del projecte (Clarisó Viladrosa, 2013)

La correspondència entre aquest model i el pla docent del TFM (Figura 6) és gairebé perfecta (Clarisó Viladrosa, 2013) (Taula 1).

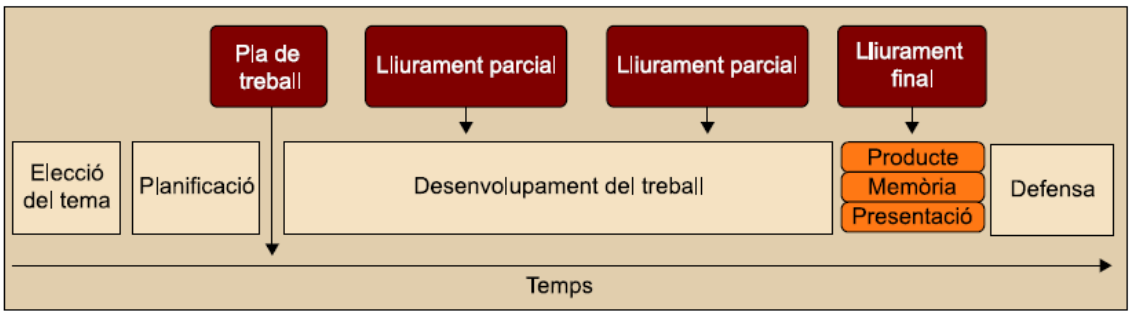


Figura 6. Etapes de la realització del treball final de màster (Clarisó Viladrosa, 2013)

Iniciació	Elecció del tema de treball
Planificació	Pla de treball
Execució	Desenvolupament del treball
Seguiment i control	Informes de seguiment, lliuraments de les PACs i retroaccions de l'equip docent
Tancament	Lliurables finals

Taula 1. Relació entre el cicle de gestió de projectes i el treball final de màster, adaptat de (Clarisó Viladrosa, 2013)

1.4.1 Abast del Treball Final de Màster (TFM)

Què inclou aquest TFM?

- L'estudi, a grans trets, de l'estat de l'art del control autònom de la locomoció bípeda mitjançant agents intel·ligents, tot identificant els diferents paradigmes de la intel·ligència artificial que utilitzen actualment per resoldre el problema, els problemes concrets per als quals són de més utilitat en el camp d'estudi en qüestió, el grau de complexitat de la seva implementació i la idoneïtat o no d'utilitzar-los en aquest TFM.

- El desenvolupament d'una solució d'agent intel·ligent per al control autònom de la locomoció bípeda per als models *bipedal walker* de la plataforma OpenAI Gym, inicialment en un model bidimensional (2D) i, posteriorment, l'escalat del model a un entorn tridimensional (3D).

- L'aplicació en aquesta solució de diferents paradigmes de forma integrada (holística), tot comparant els resultats de l'aplicació individual de cada paradigma amb els resultats de la seva aplicació conjunta.

- L'execució de la solució en diferents entorns (sense obstacles i amb obstacles de diferents tipus) i l'anàlisi dels resultats obtinguts.

- Els manuals d'instal·lació i d'usuari de la solució obtinguda.

Què no inclou aquest TFM?

- No és objecte d'aquest TFM un anàlisi exhaustiu de l'estat de l'art d'aquest problema, ja que probablement no hi hauria temps suficient en el marc del TFM i, a més a més, alguns dels coneixements del camp poden ser força complexes a nivell matemàtic en relació al grau de coneixements que s'espera haver assolit a través del Màster en Enginyeria Informàtica al qual pertany aquest TFM.

- Tampoc és objecte d'aquest TFM una solució viable a nivell físic, atès el poc temps que hi ha per desenvolupar-la. Només s'utilitzaran models simplificats com una possible aproximació a aquest problema.

1.4.2 Descomposició del treball en EDTs i fites

En aquest apartat es presenten la relació d'**Estructures de Definició del Treball (EDTs)**[5] en què s'ha descompost el treball, així com la relació de **Fites** (*milestones*)[6] més rellevants.

Relació d'Estructures de Definició del Treball (EDTs)	Relació de Fites més rellevants
1. Iniciació 1.1. Recerca de consultor-tutor 1.2. Elecció del tema 1.3. Redacció de la proposta inicial de TFM **2. Planificació** 2.1. Refinament inicial proposta 2.2. Descomposició inicial en Fites i EDTs 2.3. Temporització de les Fites i EDTs 2.4. Anàlisi i gestió dels riscos 2.5. Planificació dels lliuraments i la memòria **3. Execució** 3.1. Marc contextual 3.1.1. Recerca i anàlisi dels enfocaments actuals 3.1.2. Comprensió i coneixement dels paradigmes de l'RL i del DRL 3.1.3. Comprensió i coneixement de la plataforma OpenAI Gym 3.2. Elecció i justificació dels paradigmes a implementar 3.3. Identificació dels requisits 3.3.1. Requisits funcionals 3.3.2. Requisits no funcionals 3.4. Desenvolupament de la solució en un entorn 2D 3.4.1. Anàlisi i disseny de la solució 3.4.2. Implementació de la solució 3.4.3. Execució, ajustament i recopilació de dades 3.4.4. Anàlisi dels resultats 3.5. Escalat de la solució 2D a un entorn 3D 3.5.1. Anàlisi i disseny de l'escalat 3.5.2. Implementació de la solució 3.5.3. Execució, ajustament i recopilació de dades 3.5.4. Anàlisi dels resultats **4. Tancament** 4.1. Redacció de la memòria 4.2. Elaboració de la presentació 4.3. Defensa pública **5. Seguiment i control** 5.1. Lectura i anàlisi del pla d'estudis i de la temporització 5.2. Revisió de la proposta inicial 5.3. Revisió de la planificació 5.4. Redacció de l'informe de seguiment de la PAC2 5.5. Revisió del projecte i la seva planificació 5.6. Redacció de l'informe de seguiment de la PAC3 5.7. Revisió del producte final, de l'anàlisi i les conclusions 5.8. Revisió de la memòria	**PAC0** 1. Inici del TFM 2. Consultor-tutor assignat 3. Tema escollit 4. Proposta inicial de TFM (PAC0) lliurada **PAC1** 5. Abast establert 6. Retroacció de la PAC0 rebuda 7. Proposta inicial de TFM revisada 8. Diagrama de Gantt elaborat 9. Pla de treball (PAC1) lliurat **PAC2** 10. Retroacció de la PAC1 rebuda 11. Pla de treball revisat 12. Estudi de l'estat de l'art completat 13. Paradigmes d'interès escollits 14. Requisits per a la solució identificats 15. Disseny de la solució 2D completat 16. Implementació de la solució 2D finalitzada 17. Paràmetres d'execució ajustats 18. Execucions completades i dades recopilades 19. Anàlisi dels resultats finalitzat 20. Lliurable per a l'entorn 2D empaquetat 21. Informe de seguiment finalitzat 22. Lliurament de la PAC2 efectuat (informe + lliurable) **PAC3** 23. Retroacció de la PAC2 rebuda 24. Pla de treball revisat 25. Disseny de la solució 3D completat 26. Implementació de la solució 3D finalitzada 27. Paràmetres d'execució ajustats 28. Execucions completades i dades recopilades 29. Anàlisi dels resultats finalitzat 30. Lliurable per a l'entorn 3D empaquetat 31. Informe de seguiment finalitzat 32. Lliurament de la PAC3 efectuat (informe + lliurable) **PAC4** 33. Retroacció de la PAC3 rebuda 34. Revisió del producte final, de l'anàlisi de les dades i les conclusions 35. Memòria lliurada 36. Retroacció rebuda **PAC5a** 37. Presentació lliurada 38. Retroacció rebuda **PAC5b** 39. Defensa efectuada 40. Retroacció rebuda

[5] Una EDT és un conjunt de tasques o activitats que s'han de dur a terme.
[6] Una fita representa la consecució d'un objectiu o activitat rellevant.

1.4.3 Temporització

1.4.3.1 Terminis i dates clau a tenir presents

La Taula 2 mostra calendari de lliuraments de les PACs, que caldrà tenir presents en la temporització de les diferents EDTs i fites que formen part del projecte.

Activitats avaluables	Inici	Lliurament	Qualificació
PAC0 - Definició dels continguts del treball	16/09/2020	28/09/2020	05/10/2020
PAC1 - Pla de treball	29/09/2020	13/10/2020	20/10/2020
PAC2 - Desenvolupament del treball - Fase 1	14/10/2020	16/11/2020	23/11/2020
PAC3 - Desenvolupament del treball - Fase 2	17/11/2020	14/12/2020	21/12/2020
PAC4 - Redacció de la memòria	15/12/2020	05/01/2021	12/01/2021
PAC5a - Elaboració de la presentació	06/01/2021	10/01/2021	25/01/2021
PAC5b - Defensa pública	13/01/2021	20/01/2021	25/01/2021

Taula 2. Terminis i dates clau en el pla docent del TFM per al semestre 2020-1

1.4.3.2 Calendari i horari de treball

Segons s'indica als criteris d'avaluació del TFM, la dedicació de l'alumnat hauria de ser al voltant de les 300 hores.

Per al càlcul de la dedicació horària es tindrà present que el nombre d'hores disponibles setmanals serà de 18, que multiplicat per les 18 setmanes que dura el TFM es correspondria a un total de 324 hores, ajustat a termini amb un 8% de marge.

Aquestes 18 hores, a efectes de la planificació, es prorratejaran a una ràtio de 3 hores al dia, de dilluns a dissabte, deixant el diumenge com a dia de descans.

D'altra banda, tot i que els dies festius poden servir per accelerar aspectes que han quedat endarrerits, no es tindran en compte de cara a la planificació com a dies hàbils, ja que també són necessaris períodes de descans i d'atenció a les obligacions familiars. En tot cas, quedaran disponibles com a dies per a possibles contingències.

Així doncs, finalment, descomptant del període del TFM els diumenges, els festius i algunes dates senyalades en l'àmbit familiar, el nombre total de dies disponible per al TFM és de 101, que multiplicat per 3 hores al dia es correspon a un total de 303 hores, dins del marge establert.

1.4.3.3 Temporització de les fites i EDTs

D'acord al calendari de treball, als terminis i dates claus establerts, i tenint presents les interrelacions existents entre EDTs entre sí i amb les fites, s'ha fet una estimació de la durada de cadascuna de les tasques, que es resumeix en la Taula 3.

ID	Task Name	Duration	Start	Finish	Predecessors
1	inici del TFM	0 days	Wed 16/9/20	Wed 16/9/20	
2	**Iniciació**	**11 days**	**Wed 16/9/20**	**Mon 28/9/20**	**1**
3	Recerca de consultor-tutor	5 days	Wed 16/9/20	Mon 21/9/20	
4	Consultor-tutor assignat	0 days	Mon 21/9/20	Mon 21/9/20	3
5	Elecció del tema	7 days	Wed 16/9/20	Wed 23/9/20	
6	Tema escollit	0 days	Thu 24/9/20	Thu 24/9/20	5
7	Redacció de la proposta inicial de TFM	3 days	Thu 24/9/20	Sat 26/9/20	6
8	Prosta inicial de TFM lliurada (PAC0)	0 days	Mon 28/9/20	Mon 28/9/20	4;7
9	**Planificació**	**12 days**	**Tue 29/9/20**	**Tue 13/10/20**	
10	Refinament de la proposta inicial	2 days	Tue 29/9/20	Wed 30/9/20	8
11	Abast establert	0 days	Wed 30/9/20	Wed 30/9/20	10
12	Descomposició en Fites i EDTs	5 days	Thu 1/10/20	Tue 6/10/20	11
13	Temporització de les Fites i EDTs	4 days	Wed 7/10/20	Sat 10/10/20	12
14	Anàlisi i gestió dels riscos	2 days	Fri 9/10/20	Sat 10/10/20	13FS-2 days
15	Diagrama de Gantt elaborat	0 days	Sat 10/10/20	Sat 10/10/20	13;14
16	Planificació dels lliuraments i la memòria	1 day	Tue 13/10/20	Tue 13/10/20	15
17	Pla de treball lliurat (PAC1)	0 days	Tue 13/10/20	Tue 13/10/20	16
18	**Execució**	**51 days**	**Wed 14/10/20**	**Sat 12/12/20**	**9**
19	**Marc contextual**	**10 days**	**Wed 14/10/20**	**Sat 24/10/20**	
20	Recerca i anàlisi dels enfocaments actuals	10 days	Wed 14/10/20	Sat 24/10/20	
21	Comprensió i coneixement dels paradigmes de l'RL i del DRL	10 days	Wed 14/10/20	Sat 24/10/20	
22	Comprensió i coneixement de la plataforma OpenAI Gym	4 days	Wed 21/10/20	Sat 24/10/20	21FF
23	Estudi de l'estat de l'art finalitzat	0 days	Sat 24/10/20	Sat 24/10/20	20;21;22
24	Elecció i justificació dels paradigmes a implementar	1 day	Mon 26/10/20	Mon 26/10/20	23
25	Paradigmes d'interès escollits	0 days	Mon 26/10/20	Mon 26/10/20	24
26	**Identificació dels requisits de la solució a implementar**	**2 days**	**Tue 27/10/20**	**Wed 28/10/20**	
27	Requisits funcionals	2 days	Tue 27/10/20	Wed 28/10/20	23;25
28	Requisits no funcionals	2 days	Tue 27/10/20	Wed 28/10/20	23;25
29	Requisits identificats	0 days	Wed 28/10/20	Wed 28/10/20	27;28
30	**Desenvolupament de la solució en un entorn 2D**	**15 days**	**Thu 29/10/20**	**Sat 14/11/20**	
31	Anàlisi i disseny de la solució	5 days	Thu 29/10/20	Tue 3/11/20	29
32	Implementació de la solució	5 days	Wed 4/11/20	Mon 9/11/20	31
33	Execució, ajustament i recopilació de dades	5 days	Tue 10/11/20	Sat 14/11/20	32
34	Anàlisi dels resultats	5 days	Tue 10/11/20	Sat 14/11/20	33SS
35	Lliurable per a l'entorn 2D empaquetat	0 days	Sat 14/11/20	Sat 14/11/20	33
36	**Escalat de la solució 2D a un entorn 3D**	**23 days**	**Mon 16/11/20**	**Sat 12/12/20**	**30**
37	Anàlisi i disseny de l'escalat	6 days	Mon 16/11/20	Sat 21/11/20	
38	Implementació de la solució	8 days	Mon 23/11/20	Tue 1/12/20	37
39	Execució, ajustament i recopilació de dades	9 days	Wed 2/12/20	Sat 12/12/20	38
40	Anàlisi dels resultats	9 days	Wed 2/12/20	Sat 12/12/20	39SS
41	Lliurable per a l'entorn 3D empaquetat	0 days	Sat 12/12/20	Sat 12/12/20	39
42	**Tancament**	**26 days**	**Tue 15/12/20**	**Wed 20/1/21**	
43	Redacció de la memòria	14 days	Tue 15/12/20	Tue 5/1/21	18;59
44	Elaboració de la presentació	9 days	Mon 28/12/20	Sat 9/1/21	43FS-6 days
45	Defensa pública	7 days	Wed 13/1/21	Wed 20/1/21	44
46	**Seguiment i control**	**105 days**	**Wed 16/9/20**	**Mon 25/1/21**	
47	Lectura i anàlisi del pla d'estudis	1 day	Wed 16/9/20	Wed 16/9/20	1
48	Lliurament de la PAC0 efectuat (proposta inicial)	0 days	Mon 28/9/20	Mon 28/9/20	8
49	Retroacció de la PAC0 rebuda	0 days	Mon 5/10/20	Mon 5/10/20	48
50	Revisió de la proposta inicial de TFM	1 day	Mon 5/10/20	Mon 5/10/20	49
51	Lliurament de la PAC1 efectuat (planificació)	0 days	Tue 13/10/20	Tue 13/10/20	17
52	Retroacció de la PAC1 rebuda	0 days	Tue 20/10/20	Tue 20/10/20	51
53	Revisió de la planificació	1 day	Tue 20/10/20	Tue 20/10/20	52
54	Redacció de l'informe de seguiment de la PAC2	3 days	Fri 13/11/20	Mon 16/11/20	34FF+1 day
55	Lliurament de la PAC2 efectuat (informe + lliurables)	0 days	Mon 16/11/20	Mon 16/11/20	54
56	Retroacció de la PAC2 rebuda	0 days	Mon 23/11/20	Mon 23/11/20	55
57	Revisió del projecte i la seva planificació	5 days	Mon 23/11/20	Fri 27/11/20	56
58	Redacció de l'informe de seguiment de la PAC3	3 days	Fri 11/12/20	Mon 14/12/20	40FF+1 day
59	Lliurament de la PAC3 efectuat (informe + lliurables)	0 days	Mon 14/12/20	Mon 14/12/20	58
60	Retroacció de la PAC3 rebuda	0 days	Mon 21/12/20	Mon 21/12/20	59
61	Revisió del producte final, de l'anàlisi i les conclusions	6 days	Mon 21/12/20	Wed 30/12/20	60
62	Lliurament de la PAC4 efectuat (memòria)	0 days	Tue 5/1/21	Tue 5/1/21	43
63	Retroacció de la PAC4 rebuda	0 days	Tue 12/1/21	Tue 12/1/21	62
64	Revisió de la memòria	1 day	Tue 12/1/21	Tue 12/1/21	63
65	Lliurament de la PAC5a efectuat (presentació)	0 days	Sat 9/1/21	Sat 9/1/21	44
66	Retroacció de la PAC5a rebuda	0 days	Mon 25/1/21	Mon 25/1/21	65
67	Lliurament de la PAC5b efectuat (defensa pública)	0 days	Wed 20/1/21	Wed 20/1/21	45
68	Retroacció de la PAC5b rebuda	0 days	Mon 25/1/21	Mon 25/1/21	67

Taula 3. Relació d'EDTs i fites[7], amb la durada, dates d'inici i fi, i interrelacions de cadascuna. (taula elaborada amb el programari Microsoft Project)

[7] Les tasques de durada 0 dies es corresponen a fites (*milestones*)

A partir d'aquestes dades s'ha generat el diagrama de Gantt que es mostra en la Figura 7 i que resumeix gràficament la planificació del TFM.

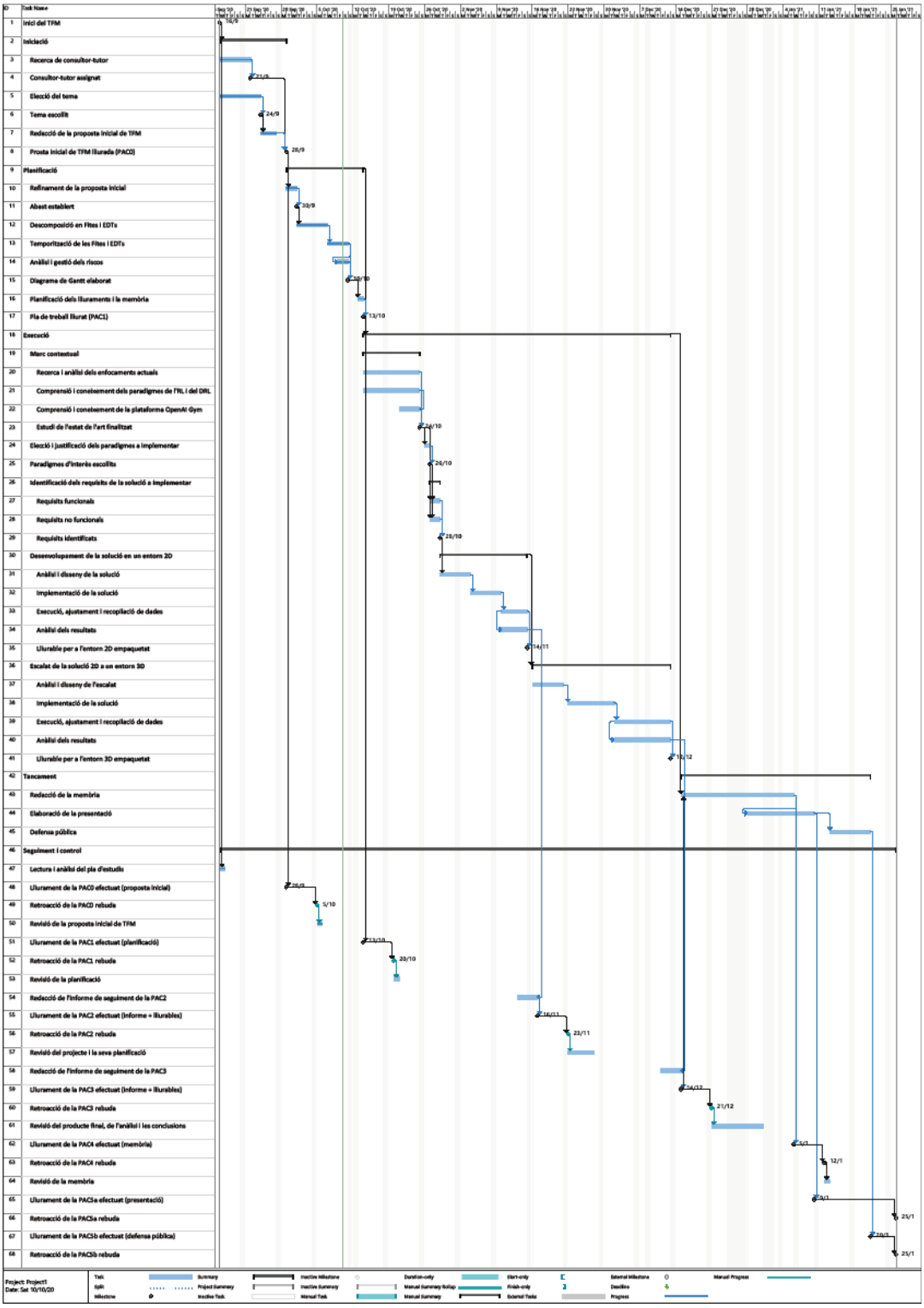


Figura 7. Diagrama de Gantt del TFM (diagrama elaborat amb el programari Microsoft Project)

Atès que la visibilitat del diagrama, un cop incrustat en aquest document, és força dolenta (per la gran quantitat d'elements que hi intervenen), es proporciona també una còpia d'aquest diagrama en format A3 en un fitxer PDF adjunt a aquesta memòria.

1.4.4 Anàlisi i gestió de riscos

1.4.4.1 Identificació dels riscos potencials

Durant la planificació es van identificar les següents situacions que podrien posar la consecució del TFM en risc:

1. Retard en l'estudi de l'estat de l'art per excés d'informació (infoxicació)
2. Complexitat en la proposta de solució
3. Retard en el desenvolupament de les solucions
4. Temps d'execució massa elevats, retard en l'obtenció de resultats
5. Entorn 3D poc desenvolupat/documentat

1.4.4.2 Anàlisi qualitativa dels riscos

Els riscos es van valorar d'acord a dues dimensions:

- La **probabilitat** que es materialitzin
- L'**impacte** que tindrien per al TFM

El **nivell de risc global** per a cada risc identificat es va obtenir d'acord a la relació que es mostra en la Taula 4.

		Impacte					
		Nul	**Molt baix**	**Baix**	**Moderat**	**Alt**	**Molt alt**
Probabilitat	**Nul·la**	*Nul*	*Nul*	*Nul*	*Nul*	*Nul*	*Nul*
	Molt baixa	*Nul*	Molt baix	Molt baix	Baix	Moderat	Moderat
	Baixa	*Nul*	Molt baix	Baix	Moderat	Moderat	Alt
	Moderada	*Nul*	Baix	Moderat	Moderat	Alt	Alt
	Alta	*Nul*	Baix	Moderat	Alt	Alt	Molt alt
	Molt alta	*Nul*	Moderat	Alt	Alt	Molt alt	Molt alt

Taula 4. Escales de valoració del nivell de risc

El resultat obtingut d'aplicar aquesta metodologia als riscos identificats és el que es mostra en la Taula 5.

Risc		**Probabilitat**	**Impacte**	**Nivell de risc**
1	Retard en l'estudi de l'estat de l'art per excés d'informació (infoxicació)	Moderada	Moderat	**Moderat**
2	Complexitat en la proposta de solució	Moderada	Alt	**Alt**
3	Retards en el desenvolupament de les solucions	Moderada	Alt	**Alt**
4	Temps d'execució massa elevats, retard en l'obtenció de resultats	Alta	Molt alt	**Molt alt**
5	Entorn 3D poc desenvolupat/documentat	Baixa	Alt	**Moderat**

Taula 5. Valoració qualitativa dels riscos identificats

Atès que tots els riscos identificats són de nivell moderat, alt o molt alt, tots ells hauran de ser gestionats.

1.4.4.3 Gestió dels riscos

La gestió de riscos consisteix en l'aplicació de mesures de correcció o mitigació dels riscos actuals per tal de disminuir-ne el seu nivell a un estat assumible que s'anomena **nivell residual**.

La Taula 6 detalla les principals mesures per a la gestió cadascun dels riscos identificats, així com el nivell de risc residual que es va decidir assumir.

Risc		Accions de correcció / mitigació	Risc residual
1	Retard en l'estudi de l'estat de l'art per excés d'informació (infoxicació)	• Per accelerar la recollida d'informació, identificar els paràmetres més rellevants de les diferents metodologies i fer el recull d'informació de forma tabular. • Després de la data límit prevista en la planificació, les metodologies que no s'hagin pogut recollir no seran considerades en aquest treball (aquest fet pot afectar la qualitat del TFM, però permet garantir el compliment de la planificació). Si les metodologies recollides es consideren insuficients, es pot valorar ampliar uns dies aquesta tasca.	**Baix**
2	Complexitat en la proposta de solució	• Ser prudent en la proposta de la solució, és preferible proposar menys i viable. • La complexitat d'implementació serà un dels paràmetres a recollir en l'estat de l'art, fet que permetrà classificar també els escenaris en funció de la complexitat i escollir aquells que entrin dins els paràmetres temporals del TFM.	**Baix**
3	Retards en el desenvolupament de les solucions	• Planificar el TFM sense utilitzar diumenges i festius deixa aquests dies per a possibles contingències; en cas de necessitat caldrà utilitzar-los. • En cas que el problema sigui degut a la complexitat de la solució proposada, valorar la possibilitat d'implementar una alternativa menys complexa d'entre les valorades. • És molt important detectar els problemes el més aviat possible i prendre les mesures adients a temps, altrament, la planificació es pot veure seriosament afectada.	**Baix**
4	Temps d'execució massa elevats, retard en l'obtenció de resultats	• Si els temps d'execució són massa elevats, caldrà centrar els esforços en la solució 2D, optimitzant al màxim els algorismes o, si convé, replantejant-se si cal canviar els algorismes escollits per altres menys complexes. Caldria aleshores renegociar amb el tutor l'abast del projecte i potser reduir-lo només al cas 2D, amb una proposta d'aplicació per al cas 3D.	**Baix**
5	Entorn 3D poc desenvolupat/documentat	• En la recerca preliminar efectuada durant la redacció de la proposta inicial de TFM, s'ha identificat treballs elaborats en models de *walkers* 3D de *pyBullet* associats a models desenvolupats a la plataforma OpenAI Gym, de manera que probablement hi haurà prou documentació i models per dur a terme el seu desenvolupament. En el cas que no sigui així, es pot obtenir una llicència gratuïta de prova per a l'entorn MuJoCo, que és el que ve configurat per defecte amb l'OpenAI Gym. Aquest temps de llicència seria suficient per al desenvolupament de la solució 3D en el marc del TFM.	**Molt baix**

Taula 6. Mesures per a la gestió dels riscos identificats i nivell de risc residual que s'assumirà

Un cop obtinguts aquests elements de gestió dels riscos, es va revisar la planificació per tal de garantir que es tenien presents.

1.4.4.4 Desviacions patides en relació a la planificació i accions de mitigació efectuades

Al llarg de la PAC2

Malauradament, degut la complexitat en l'estudi de l'estat de l'art, no es va poder iniciar el desenvolupament de la solució 2D en aquest període.

Malgrat s'havia previst una captació d'informació en forma tabular sobre els diferents enfocaments de la disciplina en estudi, l'elevat nombre de característiques dels diferents mètodes i la complexitat matemàtica d'alguns dels algorismes va portar a l'estudiant a considerar com a més adequat dur a terme una descripció textual de cada mètode per tal de facilitar-ne la comprensió.

Així mateix, d'acord al pla de gestió de riscos, es va ampliar el temps de l'estudi de l'estat de l'art, ja que es considerava que el fet d'aprofundir en els coneixements de la disciplina del *Reinforcement Learning* i del *Deep Reinforcement Learning* podia aportar molt més valor a la solució final.

Amb aquest contratemps es feia inviable acabar complint amb tots els objectius inicialment plantejats, de manera que, tal i com es preveia en la gestió de riscos, es va sol·licitar una revisió de l'abast del projecte per tal d'eliminar la part del desenvolupament de la solució 3D, que només s'acabaria abordant en el cas que el desenvolupament de la solució 2D fos més àgil de l'esperat i acabés sobrant temps.

Amb la proposta de modificació de l'abast també es va actualitzar el cronograma i el diagrama de Gantt corresponent.

Al llarg de la PAC3

Malgrat es va poder lliurar un producte més o menys elaborat, faltava encara acabar de documentar el codi, generar el manual d'ús i l'API del producte, així com executar encara alguns jocs de prova addicionals. Aquestes tasques es va proposar dur-les a terme en paral·lel durant la redacció de la memòria, tal i com s'havia previst en la planificació en l'EDT *5.7 Revisió del producte final, de l'anàlisi i les conclusions*.

Així mateix, en la planificació es preveia intentar integrar diferents metodologies entre sí. Al no haver resultat satisfactori l'agent NAF implantat, no va ser possible dur a terme aquesta integració (per exemple amb el paradigma del *Safe Reinforcement Learning*), amb la qual cosa es va haver de deixar com una proposta de futur, com es veurà més endavant en la memòria.

També alguns requisits no funcionals no es van acabar d'implementar, com es veurà, però aquests aspectes es tractaran en l'apartat corresponent de la memòria.

Tots els aspectes no coberts i no abastables durant la fase de redacció de la memòria es va proposar d'eliminar-los de l'abast del projecte, i així ho va acceptar el tutor del TFM.

1.5 Breu sumari de productes obtinguts

1.5.1 Producte obtingut

El producte obtingut és un complement per a la biblioteca TF-Agents. Incorpora agent que implementa l'algorisme NAF, les utilitats necessàries per al seu funcionament i un *wrapper* per a l'entorn per compactar diversos *time steps*.

La seva estructura en mòduls i classes és la següent:

- Mòdul **agents**
  - Mòdul **naf**
    - Mòdul **naf_agent.py**
      - Classe *NafAgent*
    - Mòdul **utils.py**
      - Classe *CompositeNetwork*
      - Classe *CompositeValueNetwork*
      - Classe *CompositeLNetwork*
      - Classe *CompositeActorDistributionNetwork*
- Mòdul **environments**
  - Mòdul **wrappers.py**
    - Classe *ActionRepeatHistoryWrapper*
- Mòdul **experiments**
  - Mòdul **off_p_tsb_experiment.py**
    - Classe *OffPolicyTimeStepBasedExperiment*
  - Mòdul **rnd_experiment.py**
    - Classe *RndExperiment*
  - Mòdul **sac_experiment.py**
    - Classe *SacExperiment*
  - Mòdul **naf_experiment.py**
    - Classe *NafExperiment*
- Mòdul **main.py**
- LICENSE.txt

1.5.2 Requisits del sistema

Les proves s'han executat amb fluïdesa en un portàtil amb el següent hardware:

- Processador de 4 nuclis Intel® Core™ i7-6700HQ 2.6GHz
- Targeta gràfica: NVIDIA GeForce GTX 950M (però no se n'ha pogut fer ús)
- 8 GB RAM

1.5.3 Altres elements lliurats

També formen part del lliurament final del TFM els següents elements:

- L'**API** del producte, com a documentació per a l'usuari/programador, en html.
- Vídeos més rellevants de les execucions de les polítiques generades
- Traces d'algunes de les execucions (es poden visualitzar amb TensorBoard)

1.6 Breu descripció dels altres capítols de la memòria

Capítol 2: Caracterització del problema

Es descriu amb detall el problema plantejat i es relaciona amb la disciplina de la intel·ligència artificial que el permet resoldre: l'aprenentatge per reforç.

Capítol 3: Estat de l'art de l'Aprenentatge per Reforç

S'analitzen els fonaments teòrics d'aquesta disciplina i els principals mètodes d'aplicació al problema a resoldre.

Capítol 4: La Plataforma OpenAI Gym

S'analitza la plataforma OpenAI Gym i els entorns que ofereix per investigar solucions per al problema proposat.

Capítol 5: Proposta de solució a desenvolupar

Es proposa una metodologia per implementar un agent que doni solució al problema i se n'estableixen les especificacions.

Capítol 6: Anàlisi i disseny de la solució

S'analitzen les diferents biblioteques disponibles per a *Reinforcement Learning* i se n'escull una (la biblioteca TF-Agents). S'escull l'algorisme a implementar (l'algorisme NAF), i es dissenya el producte a desenvolupar.

Capítol 7: Implementació en Python

Aspectes més rellevants de la implementació del producte en Python.

Capítol 8: Resultats

Recull dels resultats obtinguts més rellevants, amb una breu discussió per a cadascun d'ells.

Capítol 9: Conclusions

Conclusions en referència al treball desenvolupat, al seguiment dels objectius i de la planificació, i propostes de línies de treball futures.

Capítol 10: Glossari

Breu glossari dels termes més rellevants

Capítol 11: Bibliografia

Bibliografia referenciada al llarg de la memòria.

Capítol 12: Annexos

Codi dels mòduls i classes que no s'ha pogut cobrir en el capítol 7.

2 Caracterització del problema

2.1 Enunciat del problema

Obtenir agent intel·ligent per al control autònom de la locomoció bípeda d'un robot simulat en un entorn simulat d'acord a un objectiu prefixat.

En el cas d'aquest TFM, l'objectiu prefixat serà el d'avançar en una direcció i sentit determinats el més ràpidament possible tot superant diferents obstacles i minimitzant el risc de caiguda.

2.2 Inputs que rebrà l'agent

Es tracta d'un espai continu d'observacions de la forma $\mathbb{R}^n$, on n és el número de mesures que rep l'agent en cada observació, tant de l'entorn com del propi robot.

Aquests inputs es poden classificar en:

- Informació de l'estat del robot:
  - Per exemple, mesures sobre la inèrcia dels seus efectors
- Informació parcial sobre l'entorn:
  - Per exemple, mitjançant sensors LIDAR
- Informació sobre l'efectivitat instantània de la política d'actuació:
  - Una mesura en forma d'un escalar $r \in \mathbb{R}$, anomenada *recompensa*, que indica l'èxit de l'agent en l'entorn d'acord als criteris establerts

2.3 Outputs que generarà l'agent

- Intensitat i sentit a aplicar a cadascun dels efectors del robot:

  Es tracta d'un espai continu d'accions (discretitzable, si convé) de forma $\mathbb{R}^m$, on m és el número d'efectors, i on cada component es troba acotada per a les intensitats màximes permeses per a cada efector.

2.4 Disciplina de la Intel·ligència Artificial en què s'encabeix

En aquest cas, l'agent aprèn el seu comportament directament a partir la seva interacció amb l'entorn, és a dir, d'allò que percep i de les conseqüències d'aquelles accions que decideix executar. Aquest tipus d'aprenentatge encaixa dins la disciplina de l'**Aprenentatge per Reforç** (*Reinforcement Learning*).

"A diferència de les situacions d'aprenentatge automàtic «clàssiques», en les quals hi ha un conjunt de dades, etiquetatges o no, amb els quals s'entrena un sistema, en ***l'aprenentatge per reforç*** *un sistema aprèn per la seva interacció amb l'entorn, concretament per la situació que percep, les accions que executa i les conseqüències de les seves accions."* (Benítez, Escudero, Kanaan, Masip Rodó, & Cencerrado Barraqué, 2018)

Així mateix, en els darrers anys, aquesta disciplina ha experimentat una important evolució amb l'ús de les metodologies d'aprenentatge profund (*Deep Learning*), donat lloc a la disciplina del ***Deep Reinforcement Learning***.

3 Estat de l'art de l'Aprenentatge per Reforç

3.1 Marc conceptual de l'Aprenentatge per Reforç

3.1.1 Elements de l'Aprenentatge per reforç

En l'**aprenentatge per reforç** (en endavant **RL**, de *Reinforcement Learning*), un **agent intel·ligent** interacciona amb el seu **entorn** $\boldsymbol{E}$ al llarg del temps (Figura 8). En aquestes interaccions, l'agent observa l'entorn i hi du a terme accions, les quals tenen conseqüències que és capaç de percebre. A partir d'aquestes percepcions, l'agent va aprenent com comportar-se, tot relacionant aquelles situacions en què es va trobant amb aquelles accions que és més probable que maximitzin el seu èxit en l'entorn en qüestió.

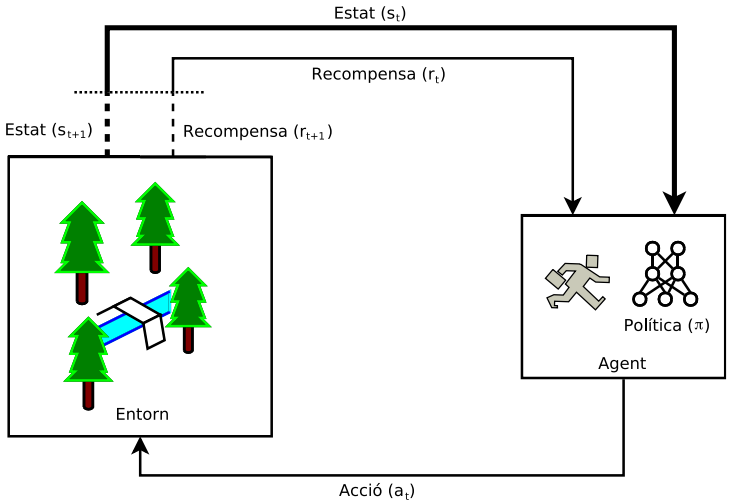


Figura 8. Elements de l'Aprenentatge per Reforç

3.1.1.1 L'espai de temps $\mathcal{T}$

Malgrat el temps té una naturalesa contínua, l'agent intel·ligent sol executar-hi les seves accions en un **espai de temps discretitzat** $\mathcal{T}$, de manera que les interaccions de l'agent amb l'entorn es produeixen en instants concrets t, d'acord a una freqüència de discretització determinada. L'interval de temps entre un instant t i l'instant següent $t + 1$ es coneix com a *time step*.

3.1.1.2 L'espai d'estats $\mathcal{S}$

En cada instant t, l'agent rep (o percep) un conjunt de valors que descriuen l'**estat** $\boldsymbol{s_t}$ en què es troben conjuntament entorn i agent[8]. El conjunt de tots els estats possibles per a qualsevol t rep el nom d'**espai d'estats** $\mathcal{S}$. Aquest espai pot ser discret o continu, segons la naturalesa de les propietats percebudes.

[8] Alguns autors fan distinció entre l'estat de l'entorn i l'estat de l'agent, de manera que l'estat de l'agent engloba una percepció d'algunes característiques l'entorn i d'algunes característiques del propi agent. També és comú definir una funció ϕ que retorna l'estat s_t en qüestió a partir d'una observació O_t efectuada per l'agent; aquesta funció pot realitzar tractaments previs, reducció de dimensionalitat, addició d'informació de l'estat intern del propi agent, etc.

3.1.1.3 L'espai d'accions $\mathcal{A}$

En aquest moment, l'agent seleccionarà i executarà una **acció a_t** en l'entorn en què es troba, de manera que agent i entorn transicionaran conjuntament cap al **següent estat s_{t+1}**. A la vegada, l'agent rebrà un *feedback* de l'entorn en forma d'un valor escalar anomenat **recompensa (*reward*) r_{t+1}**.

El conjunt de totes les accions possibles que pot escollir l'agent per a qualsevol estat s i instant t rep el nom d'**espai d'accions $\mathcal{A}$**. Aquest espai pot ser tant discret com continu, segons la naturalesa de les accions que es poden executar.

3.1.1.4 Les transicions i la funció de probabilitat de transició $\mathcal{P}$

L'entorn doncs, es comporta com una **funció de transició** entre estats que ve determinada per una **funció de probabilitat de transició** $\mathcal{P}(s_{t+1}|s_t, a_t)$ que depèn de l'instant i estat concrets en què es troba l'agent i de l'acció que s'hi decideixi executar ($\mathcal{P}: \mathcal{S} \times (\mathcal{S} \times \mathcal{A} \times \mathcal{T}) \to \mathbb{R}$). Alguns algorismes d'RL intenten generar un model d'aquest comportament de l'entorn, de manera que poden aprendre nous comportaments fent simulacions sobre aquest model i evitar executar les accions en exploració en l'entorn real fins que no s'han avaluat prèviament mitjançant el model aprés o en aprenentatge.

3.1.1.5 Les recompenses i la funció de recompensa $\mathcal{R}$

Un cop executada l'acció a_t i transicionats agent i entorn cap al següent estat s_{t+1}, tal i com s'ha comentat, l'agent rep també un *feedback* anomenat **recompensa (*reward*) r_{t+1}** en relació a l'acció a_t executada. Aquesta recompensa té en general la forma d'un escalar, malgrat alguns mètodes defineixen aquest valor amb una aproximació vectorial amb diferents components (van Seijen, et al., 2017).

Aquest valor de recompensa pot tractar-se d'una mesura directa de l'entorn (per exemple, la distància a un determinat element o l'alçada del cap del robot respecte al terra), o també pot tractar-se d'un càlcul a partir d'aquestes mesures (per exemple una funció que determini si el robot ha caigut a terra o no). Sigui com sigui, a la funció que retorna la recompensa se l'anomena funció de recompensa $\mathcal{R}$ i normalment respon a una aplicació de tipus $\mathcal{R}: \mathcal{S} \times \mathcal{A} \times \mathcal{T} \to \mathbb{R}$.

3.1.1.6 La política d'actuació π

La funció que determina quina acció executar en cada instant per a l'estat concret en què es troba l'agent es coneix amb el nom de **política π** d'actuació. Intuïtivament es correspon a un mapatge entre l'espai d'estats i l'espai d'accions en funció del temps.

3.1.1.6.1 Polítiques deterministes vs. polítiques estocàstiques

Una política **determinista** és aquella en que l'acció a executar sempre serà la mateixa donats l'estat i instant en què es trobi l'agent. Una política **estocàstica**, en canvi, és aquella on l'acció a executar es determina a partir d'una funció de

probabilitat per a les diferents accions possibles que depèn de l'estat i instant en què es troba l'agent.

En l'exemple clàssic del pedra-paper-tisores (*tic-tac-toe*), una política determinista posaria l'agent en desavantatge, ja que la seva jugada sempre seria la mateixa donat un estat i instant concrets. En aquest escenari, una política estocàstica li permetria anar modificant la jugada aleatòriament d'acord a una funció de probabilitat que podria tenir en compte, per exemple, les costums del jugador contrari.

En el cas de les **polítiques estocàstiques**, la funció $\boldsymbol{\pi}$ se sol formalitzar de la següent manera:

$$\pi(a|s_t) = P[a_t = a|s_t] \qquad \text{Expressió 1}$$

De manera que la política respon a una aplicació del tipus $\boldsymbol{\pi}$: $\mathcal{S} \times \mathcal{T} \times \mathcal{A} \rightarrow \mathbb{R}$, on l'escalar representa la probabilitat d'escollir l'acció a en l'instant t, donat l'estat s_t.

En el cas de les **polítiques deterministes**, en canvi, la política sol representar-se amb el símbol μ i se solen formalitzar de la següent manera:

$$\mu(s_t) = a \qquad \text{Expressió 2}$$

Atès que donats un estat i instants concrets, sempre se seleccionarà la mateixa acció a executar. Aquestes funcions responen a una aplicació del tipus $\boldsymbol{\mu}$: $\mathcal{S} \times \mathcal{T} \rightarrow \mathcal{A}$.

En molts algorismes, les polítiques deterministes es consideren un cas particular de les polítiques estocàstiques, quan la probabilitat d'escollir l'acció és del 100% per a una sola acció en tots els estats possibles en qualsevol instant.

3.1.1.6.2 La funció que defineix la política d'actuació

Finalment, quan el nombre d'estats és discret, finit i suficientment abastable, així com el nombre d'instants a tenir presents (o millor encara, si la política és independent de l'instant en què es troba un estat), la funció que defineix política d'actuació es pot formalitzar de manera **tabular** (Figura 9).

$\boldsymbol{\pi(a\|s_t)}$	$\boldsymbol{s_{1,t=1}}$	$\boldsymbol{s_{2,t=1}}$	$\boldsymbol{s_{1,t=2}}$	$\boldsymbol{s_{2,t=2}}$
$\boldsymbol{a_1}$	$P[a_1\|s_{1,t=1}]$	$P[a_1\|s_{2,t=1}]$	$P[a_1\|s_{1,t=2}]$	$P[a_1\|s_{2,t=2}]$
$\boldsymbol{a_2}$	$P[a_2\|s_{1,t=1}]$	$P[a_2\|s_{2,t=1}]$	$P[a_2\|s_{1,t=2}]$	$P[a_2\|s_{2,t=2}]$
$\boldsymbol{a_3}$	$P[a_3\|s_{1,t=1}]$	$P[a_3\|s_{2,t=1}]$	$P[a_3\|s_{1,t=2}]$	$P[a_3\|s_{2,t=2}]$

Figura 9. Exemple de definició tabular de la política $\boldsymbol{\pi}$

Tanmateix, són pocs els casos en què malgrat el nombre d'estats sigui discret i finit, el problema acabi essent tractable, ja que la resolució d'aquest enfocament tabular pateix de la maledicció de la dimensionalitat. En aquests casos, o quan l'espai d'estats o d'accions és continu, la funció que defineix la política se sol formalitzar mitjançant una **funció d'aproximació parametritzada**, ja sigui **lineal**

o **no lineal** (com pot ser una **xarxa neuronal**), per aproximar el comportament de la política.

En els casos de funcions d'aproximació parametritzades, el conjunt de variables que permeten aproximar la funció a la política en qüestió rep el nom de paràmetre θ, i les polítiques queden formalitzades en referència a aquest paràmetre (Expressió 3 i Expressió 4).

$$\pi_\theta(a|s_t) = P[a_t = a|s_t; \theta] \qquad \text{Expressió 3}$$

$$\mu_\theta(s_t) = a \qquad \text{Expressió 4}$$

3.1.2 Objectiu de l'Aprenentatge per reforç

L'objectiu de l'RL, durant la fase d'aprenentatge, serà el de trobar la millor política possible, és a dir, aquella que maximitzi la recompensa a llarg termini sigui quin sigui l'estat i instant en què es trobi l'agent. Aquest objectiu, però, convé formalitzar-lo adientment en termes matemàtics per tal d'obtenir algorismes capaços de resoldre'l.

3.1.2.1 El valor de retorn descomptat acumulat a llarg termini $\boldsymbol{R_t}$

Tradicionalment, s'acostuma definir el **valor de retorn descomptat acumulat a llarg termini $\boldsymbol{R_t}$** com un escalar que representa, per a l'instant t, el valor acumulat de les recompenses futures tenint en compte un **factor de descompte $\boldsymbol{\gamma} \in (0,1]$** geomètric respecte al temps (Expressió 5), de manera que com més futura és la recompensa rebuda seguint la política en qüestió, menys importància se li dona a aquesta recompensa futura respecte a l'instant en què s'està valorant aquesta expressió.

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \qquad \text{Expressió 5}$$

En general, no és viable calcular el valor R_t per a totes les combinacions d'estats i accions possibles en un moment donat i futurs, i convé fer-ne una estimació. Les funcions que estimen aquest valor (o valors derivats d'aquest) es coneixen amb el nom de **funcions valor**, entre les quals destaquen les funcions **estat-valor (v)** i **acció-valor (q)**, com es veurà més endavant.

D'aquesta manera, l'objectiu de l'RL es pot formalitzar com la cerca de la política π que permeti maximitzar el valor estimat d'R_t per a qualsevol instant i estat en què es trobi l'agent. Aquesta política rep el nom de **política òptima $\boldsymbol{\pi^*}$**.

$$\pi^* = \arg\max_\pi \mathbb{E}[R_t|\pi] \qquad \text{Expressió 6}$$

3.1.2.2 Les funcions valor v i q

Les funcions valor són estimacions del valor R_t o de variants d'aquest valor. Les funcions valor més utilitzades són la funció **valor-estat (*V-value*)** i la funció **valor-acció (*Q-value*)**.[9]

3.1.2.2.1 Funció valor-estat (V-value) i valor òptim de l'estat

Es denomina funció **valor-estat v** de l'estat s per a la política π a aquella que retorna el valor esperat de l'escalar R_t quan l'agent es troba en l'estat s i segueix la política π.

$$v_\pi(s) = \mathbb{E}_\pi[R_t|s_t = s] \qquad \text{Expressió 7}$$

Es denomina **valor òptim v_* de l'estat s** al valor màxim que pot assolir la funció v per l'estat s, sigui quina sigui la política π que s'hagi d'utilitzar per arribar a aquest valor màxim.

$$v_*(s) = \max_\pi v_\pi(s) \qquad \text{Expressió 8}$$

Aquest valor equival al que prendria $v_\pi(s)$ en el cas de seguir la política òptima π^*, ateses les formulacions de l'Expressió 6 i de l'Expressió 7.

$$v_{\pi^*}(s) = v_*(s) \qquad \text{Expressió 9}$$

Això permet redefinir l'objectiu de l'RL mitjançant la següent expressió:

$$\pi^* = \arg\max_\pi v_\pi(s) \qquad \text{Expressió 10}$$

3.1.2.2.2 Funció valor-acció (Q-value) i valor òptim de l'acció

Es denomina funció **valor-acció q** de l'acció a en l'estat s amb política π al valor esperat de l'escalar R_t quan l'agent es troba en l'estat s, executa l'acció a i a continuació segueix la política π.

$$q_\pi(s,a) = \mathbb{E}_\pi[R_t|s_t = s, a_t = a] \qquad \text{Expressió 11}$$

Es denomina **valor òptim q_* de l'acció a en l'estat s** al valor màxim que pot assolir aquesta per a aquest estat, executant aquesta acció, sigui quina sigui la política π que s'executi a continuació.

$$q_*(s,a) = \max_\pi q_\pi(s,a) \qquad \text{Expressió 12}$$

Òbviament, atès que la política òptima és aquella que maximitza l'esperança d'R_t sigui quin sigui l'estat i instant en què es trobi l'agent, es compleix també l'expressió

$$q_*(s,a) = q_{\pi^*}(s,a) \qquad \text{Expressió 13}$$

[9] S'acostuma a referir als valors reals d'aquestes funcions mitjançant les lletres minúscules (q, v), i a les seves estimacions mitjançant lletres majúscules (Q, V).

D'altra banda, quan s'avaluï l'acció que maximitza $q_*(s,a)$, el valor que s'obté és precisament el valor òptim de l'estat $v_*(s)$, i l'acció seleccionada es correspon a una de les possibles per a la política òptima quan l'agent es troba en aquest estat.

$$\max_{\text{a}} q_*(s,a) = \max_{\text{a}} q_{\pi^*}(s,a) = v_*(s) \qquad \text{Expressió 14}$$

Això permet definir l'objectiu de l'RL com la cerca, per a cada estat i instant, d'aquelles accions que maximitzen la funció valor òptim de l'acció per aquest estat i instant. A aquesta acció se l'anomena acció òptima $a^*(s)$ per al context s_t determinat.

$$a^*(s) = \arg\max_a \ q_*(s,a) \qquad \text{Expressió 15}$$

La política òptima compleix ara amb la següent propietat:

$$\pi^* = \arg\max_\pi q_*(s,a) \qquad \text{Expressió 16}$$

3.1.2.2.3 Funció avantatge i valor òptim de l'acció

Es denomina funció **avantatge** $\boldsymbol{A}$ de l'acció a per a l'estat s_t sota la política π a la diferència entre el Q-value i el V-value que a aquesta acció i estat.

$$A_\pi(s_t,a) = Q_\pi(s_t,a) - V_\pi(s_t) \qquad \text{Expressió 17}$$

Per definició, la funció **valor-acció** també es pot expressar com:

$$q_\pi(s,a) = r_{t+1,a} + \gamma v_\pi(s_{t+1}) \qquad \text{Expressió 18}$$

Per tant, la funció avantatge també es pot expressar com:

$$A_\pi(s,a) = r_{t+1,a} + \gamma V_\pi(s_{t+1}) - V_\pi(s_t) \qquad \text{Expressió 19}$$

Alguns mètodes, com les *Dueling DQN* (Wang, et al., 2016), calculen la funció valor-estat per una banda i la funció avantatge per una altra, i aleshores les combinen per obtenir la funció valor-acció a partir d'aquestes dues. Això els permet una millor discriminació en la funció valor-acció per aquelles accions que tenen valors similars.

Altres mètodes, com alguns basats en gradients i alguns mètodes actor-crític, utilitzen la funció avantatge com una manera per reduir la variància de la solució, atès que consideren la funció valor-estat com una **línia de base** que es pot eliminar de l'expressió de l'estat-acció de manera que es dona més importància a la diferència relativa respecte a aquesta línia de base. Això els proporciona millors propietats de convergència.

3.2 Principals enfocaments de l'RL

3.2.1 La recerca de la millor política

Hi ha diferents formes d'intentar obtenir una política d'actuació. Aquestes poden partir de coneixements i models previs (*model-based*) o no (*model-free*), i se poden centrar en estimar primer una funció valor i a partir d'aquesta, trobar les accions que la maximitzen per a cada estat i instant (*value-based*), o bé cercar directament la política i veure quines conseqüències té, tot ajustant la cerca en base a aquestes (*policy-based*).

Així mateix, podem trobar multitud de metodologies híbrides, que combinen diferents d'aquestes metodologies. Els mètodes actor-crític, per exemple, combinen rutines *value-based* amb rutines *policy-based*, de manera que les primeres (el crític) guien a les segones (l'actor), com s'explicarà més endavant. Així mateix, l'actor pot ser un actor model-free i el crític, en canvi, model-based, o viceversa.

A continuació es presenta un recull dels principals enfocaments detectats en aquest estudi de l'estat de l'art per a la solució al problema de l'RL. En primer lloc es recullen un conjunt d'estratègies habituals que utilitzen diferents mètodes de l'RL i, a continuació, es presenta una possible classificació (taxonomia) dels diferents mètodes d'RL, tot explicant les principals característiques de cadascun d'ells.

3.2.2 Estratègies habituals dins els mètodes d'RL

3.2.2.1 Els Processos de Decisió de Markov

Una propietat important a l'hora de caracteritzar els diferents tipus de problemes en l'RL és la **propietat de Markov**. Es diu que un sistema compleix aquesta propietat quan el seu estat futur immediat (s_{t+1}) depenen en tot moment únicament de l'estat actual (s_t) en què es troba el sistema i de l'acció (a_t) que se li apliqui en aquest instant, sense dependre de cap altre factor del passat.

Els problemes que satisfan la propietat de Markov es poden assimilar a un **Procés de Decisió de Markov (*MDP, Markov Decission Process*)**. Aquests problemes queden definits per la tupla $(\boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{P}}, \boldsymbol{\mathcal{R}}, \boldsymbol{\gamma})$, en què $\boldsymbol{\mathcal{S}}$ és l'espai d'estats, $\boldsymbol{\mathcal{A}}$ és l'espai d'accions, $\boldsymbol{\mathcal{P}}$ és la funció de probabilitat de transició entre estats, $\boldsymbol{\mathcal{R}}$ és la funció que permet obtenir la recompensa un cop executada una acció en un determinat estat i $\boldsymbol{\gamma}$ és el factor de descompte en les recompenses al llarg del temps.

En aquests sistemes, per la propietat de Markov,

$$\boldsymbol{\mathcal{P}}(s_{t+1}|s_t, a_t) = \boldsymbol{\mathcal{P}}(s_{t'+1}|s_{t'}, a_{t'});\ \forall t, t' \in \boldsymbol{\mathcal{T}}$$

És a dir, que la funció de probabilitat de transició és independent de la dimensió temporal i depèn només de l'estat en què es troba actualment el sistema i de l'acció executada $\boldsymbol{\mathcal{P}}: \mathcal{S} \times (\mathcal{S} \times \mathcal{A}) \rightarrow \mathbb{R}$, de manera que aquesta funció de probabilitat se sol expressar com:

$$\boldsymbol{\mathcal{P}}(s'|s, a)$$

que representa la probabilitat de transició cap a s' quan el sistema es troba en l'estat s i se li aplica l'acció a.

De la mateixa manera, la funció recompensa també serà independent de la dimensió temporal $\boldsymbol{\mathcal{R}}$: $\mathcal{S}x\mathcal{A} \rightarrow \mathbb{R}$, de manera que s'expressarà com:

$$\boldsymbol{\mathcal{R}}(a, s)$$

En els casos en què els espais d'estats i d'accions són discrets i finits, si es té un coneixement complet de les funcions $\mathcal{P}$ i $\mathcal{R}$, la política òptima per aquest MDP es pot intentar trobar mitjançant els mètodes basats en la teoria de control de processos, com és el paradigma de la *Programació Dinàmica*, que utilitza un enfocament probabilístic, com es veurà més endavant. Tanmateix, aquest enfocament pateix de la maledicció de la dimensionalitat i ràpidament els problemes esdevenen intractables a mesura que l'espai d'estats i/o d'accions creixen.

Quan això passa, es pot intentar combinar aquests mètodes amb funcions d'aproximació a $\mathcal{P}$ i $\mathcal{R}$ (mètodes *Approximated Dynamic Programming o ADP*) o bé canviar de paradigma i emprar mètodes com la *Diferència Temporal* i els basats en *Monte Carlo*[10], que s'explicaran més endavant, així com diferents combinacions entre tots ells.

Finalment, per acabar de caracteritzar els diferents tipus de problemes, quan l'agent rep tota la informació sobre l'estat en què es troba es considera un MDP completament observable (***FOMDP o Fully Observable MDP***), mentre que quan només rep informació parcial sobre l'estat es considera un MDP parcialment observable (***POMPD o Partially Observable MDP***). En aquest darrer cas, és habitual que l'agent faci una estimació sobre en quin estat es troba.

3.2.2.2 Aprenentatge per episodis

S'anomena **episodi** a l'experiència completa viscuda per un agent des que inicia la seva activitat en un estat inicial fins que evoluciona a un estat considerat com a terminal (per exemple, si el robot arriba al seu destí o si cau i es considera irrecuperable), o bé fins que s'arriba a un límit temporal fixat per l'algorisme (encara que no s'hagi arribat a un estat terminal).

Els algorismes que duen a terme un **aprenentatge per episodis**, d'una banda fixen aquesta durada màxima de l'episodi, i de l'altra, un cop arriben al final d'un episodi (ja sigui per haver arribat a un estat terminal o per haver arribat al límit temporal de l'episodi), continuen el seu aprenentatge reiniciant-se en un nou estat inicial i iniciant una nova experiència, és a dir, un nou episodi.

Exemples d'algorismes amb aprenentatge per episodis són tots aquells basats en el mètode de Monte Carlo, com es veurà més endavant.

[10] Els mètodes de *Monte Carlo* permeten tractar també problemes *no Markovians*.

3.2.2.3 La recerca de la millor política: predicció i explotació vs. control i exploració

En general, els algorismes d'RL requereixen de dos processos fonamentals per arribar a una solució: predicció i control.

El procés de **predicció** consisteix a avaluar com de bona és una política, un estat, o una acció en qüestió, sovint mitjançant l'estimació de les funcions valor a partir d'aquesta política o avaluant directament la política en l'entorn.

El procés de **control**, en canvi, consisteix a modificar la política en qüestió per tal d'optimitzar-la i aproximar-la als valors òptims de retorn per als estats o les accions seleccionades.

En funció de l'algorisme, aquests processos prenen una forma o una altra, i es poden produir de forma simultània o bé alterna en el temps.

Un exemple, com es veurà més endavant, serien els algorismes d'Iteració Generalitzada de la Política (*Generalized Policy Iteration*), en què alternativament es fixa una política potencialment òptima i es calcula una estimació de V_* a partir d'aquesta (avaluació de la política o *policy evaluation*), i posteriorment es formalitza una nova política potencialment òptima a partir dels valors-estat òptims obtinguts (millora de la política o *policy improvement*). Aquest procés es repeteix iterativament fins que s'acaba convergint a una política òptima (Figura 10).

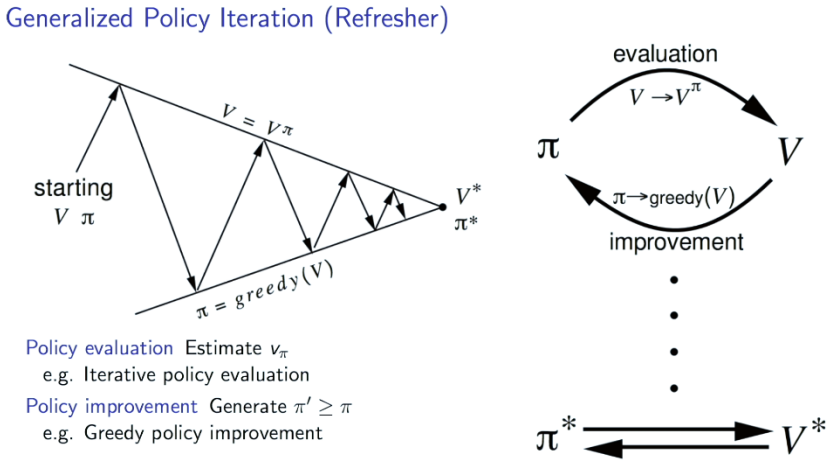


Figura 10. Algorsime d'Iteració Generalitzada de la Política, imatge extreta de (van Hasselt, 2018)

La predicció i el control tenen una influència mútua, i és que la política que s'està avaluant influeix en el resultat de les funcions valor, i aquest resultat, a la vegada, influeix en la definició de noves polítiques.

Els diferents algorismes d'RL disposen d'una política actual potencialment òptima i l'**exploten** per estimar la bondat d'aquesta (ja sigui mitjançant funcions valor o de forma directa en l'entorn o en un model d'aquest).

Tanmateix, també han d'optimitzar la política en qüestió, i per fer-ho, no sempre es disposa de tots els valors per a tots els estats i/o accions, o aquest enfocament

no és computacionalment tractable, amb la qual cosa cal que duguin a terme un procediment d'**exploració**.

Quan l'obtenció d'una política potencialment òptima es fa tenint en compte únicament les millors estimacions actuals i seleccionant l'acció que porta l'agent cap al millor estat detectat fins al moment, es diu que se segueix un procediment de millora de la política de tipus *greedy*[11] (***Greedy Policy Improvement***). Aquest procediment, però, pot convergir cap a polítiques subòptimes, ja que no permet l'exploració d'accions que no s'han explotat encara i que podrien tenir millors resultats.

Com a alternativa, un mètode habitual és el procediment ***∈-greedy***, en que s'escull la millor acció detectada fins al moment amb una probabilitat *1-∈* i, alternativament, amb una probabilitat $\in$, s'escull una acció a l'atzar. Aquest mètode permet continuar explorant tot l'espai d'accions, amb major o menor grau, a mesura que l'algorisme va convergint cap a una política òptima. Cal dir, però, que tot i que és un dels mètodes més emprats per la seva gran simplicitat, genera un cert soroll residual quan s'està ja en la solució òptima, ja que continua seleccionant accions a l'atzar d'acord a la probabilitat $\in$.

Finalment, un mètode molt més complex és el de les Cotes Superiors de Confiança (***Upper Confidence Bounds o UCB***). Aquest mètode es basa en la definició d'un nou concepte, el penediment per haver escollit una determinada acció (***regret***). Formalment es defineix el *regret* (Δ_a) com la pèrdua d'oportunitat en un *time step* pel fet d'haver seleccionat una determinada acció a. Suposant un MDP, aquest concepte quedaria definit com:

$$\Delta_a = v_* - q(a) \qquad \text{Expressió 20}$$

Això permet definir un *regret* total acumulat L_t fins a l'instant t de la següent manera:

$$L_t = \sum_{t=0}^{\infty} \big(v_* - q(a_t)\big) = \sum_a N_t(a)\big(v_* - q(a)\big) = \sum_a N_t(a)\Delta_a \qquad \text{Expressió 21}$$

L'objectiu de l'UCB és minimitzar aquest L_t i, per fer-ho, es planteja quina és el coneixement o confiança que tenim sobre l'estimació del valor-acció per cadascuna de les accions.

La clau d'aquest plantejament és que si explorem accions sobre les quals tenim força incertesa per a un estat determinat, aleshores la quantitat d'informació que obtindrem de l'entorn serà molt més rellevant que no pas la que obtindríem per aquelles accions sobre les quals ja hem fet exploracions prèvies per a aquest mateix estat.

Així doncs, el mètode defineix una funció $U_t(a,s)$ que representa el grau desconfiança respecte a l'estimació en l'instant t del Q-value ($Q_t(a,s)$). Concretament, $U_t(a,s)$ representa el valor que cal sumar a $Q_t(a,s)$ per tal

[11] Greedy es pot traduir com a golafre, voraç i/o egoista.

d'assegurar que el valor real de Q es troba dins d'aquest marge de confiança. L'Expressió 22 formalitza aquest postulat de manera que la probabilitat que aquest valor Q real sigui superior a aquesta suma sigui inferior a una probabilitat llindar determinada.

$$p\big(q(a,s) \geq Q_t(a,s) + U_t(a,s)\big) \leq llindar \qquad \text{Expressió 22}$$

Aquest valor $U_t(a,s)$ és molt elevat al principi, quan s'ha explorat mai o poques vegades l'acció a en l'estat s, i a mesura que va explorant aquesta acció en aquest estat disminueix, de manera que és més probable que el Q-valor estimat fins al moment sigui força similar al Q-valor real per a aquests estat i acció, d'acord a un llindar que en determina la confiança.

Així doncs, a mesura que incrementen el nombre d'observacions respecte a una acció a en un estat s, el valor d'$U_t(a,s)$ disminueix, de manera de confiança amb l'estimació actual del Q-value augmenta i això fa que, a no ser que tingui un Q-value elevat, aquesta acció és menys probable (però no impossible), que es torni a escollir en un futur.

Aquest mètode permet una exploració més eficient de l'espai d'accions que no pas el mètode ϵ**-greedy**, si bé és més complex d'implementar i no té un equivalent directe quan l'espai d'estats és continu i s'utilitzen funcions d'aproximació.

3.2.2.4 Aprenentatge on-policy vs. aprenentatge off-policy

Un altre enfocament habitual en els mètodes d'RL és el de fer una distinció entre la política que serveis per a l'aprenentatge i aquella que és la candidata a la política òptima.

Així doncs, alguns mètodes, parteixen d'una política base que van optimitzant fins obtenir la política òptima. Són els anomenats mètodes d'aprenentatge ***on-policiy***. Tanmateix, aquest mètode presenta diferents problemes, com són la correlació temporal entre estats i accions, i la consciència del mètode d'optimització sobre la política d'exploració (fet que pot fer que desenvolupi estratègies per evitar els riscos que comporta l'exploració).

En aquest sentit s'han proposat també mètodes ***off-policy***, que tenen una o diverses polítiques (anomenades polítiques de comportament o ***behaviour policies***) que els serveixen per anar explorant l'espai d'estats i anar estimant les funcions valor, de manera que van construint la política òptima a partir del coneixement que extreuen d'aquestes polítiques. Això permet a aquests mètodes aprendre també a partir d'exemples o reaprofitar experiències d'antigues polítiques explorades. A més a més, en poder explorar una política diferent a la que s'està construint, no és necessari fer exploracions en les accions individuals d'aquesta política, la qual cosa evitaria el biaix introduït pels mètodes d'exploració *on-policy*.
També en la literatura es troben solucions intermèdies, que utilitzen el potencial de tots dos mètodes. Així per exemple, per disminuir el biaix de les correlacions temporals, alguns mètodes utilitzen l'***experience replay***, que actualitza la funció valor en cada *time step* utilitzant alguna experiència prèvia viscuda anteriorment

escollida a l'atzar d'entre les emmagatzemades en una memòria d'experiències. Un altre exemple seria la metodologia de **l'*actor-crític***, en què una rutina de l'agent que actua com a crític, estima la funció valor (per exemple *off-policy*), i una altra rutina actua com a actor, explora l'espai d'accions tot construint una política potencialment òptima (per exemple *on-policy*), amb ajuda de la informació que li facilita el crític.

3.2.2.5 Funcions d'aproximació i el deep RL

Molts dels problemes d'RL tenen un espai d'estats i/o d'accions que són intractables mitjançant un enfocament tabular (ja sigui per l'elevat nombre d'elements de l'espai o pel fet de ser espais continus). En aquests casos les diferents funcions que s'utilitzen en l'RL (la política, les funcions valor, la funció de transició entre estats o la funció de recompensa) es poden aproximar mitjançant l'ús de **funcions d'aproximació parametritzades**.

D'aquestes funcions n'hi ha de molts tipus, tant de lineals com no lineals, així com de diferenciables i de no diferenciables. Els algorismes ajusten el paràmetre θ de la funció a mesura que aprenen sobre l'entorn. Per exemple:

$$v_\theta(s) \approx v_\pi(s) \qquad \text{Expressió 23}$$

Les aproximacions més senzilles són aquelles en què l'output és una combinació lineal de l'input, on el paràmetre θ representa el pes de cada component en aquesta combinació. En l'exemple anterior, si considerem la funció *V-value* com una combinació lineal de les característiques que es poden extreure de l'estat mitjançant la funció ϕ, aquesta quedaria expressada de la forma següent:

$$v_\theta(s) = \theta^T \phi(s) = \sum_j \phi_j(s)\,\theta_j \qquad \text{Expressió 24}$$

En aquest cas, la funció objectiu a minimitzar durant l'aproximació podria ser l'error quadràtic:

$$J(\theta) = \mathbb{E}_\pi[\left(v_\pi(s) - \theta^T \phi(s)\right)^2] \qquad \text{Expressió 25}$$

De manera que es podria minimitzar mitjançant un descens estocàstic en el gradient (*Stochastic Gradient Descent* o *SGD*) o bé mitjançant altres estratègies, com la *Least Squares Prediction*, ja sigui mitjançant mètodes com *Monte Carlo* o bé la *Temporal Difference* (*LSMC* i *LSTD*, respectivament).

Una altra forma d'aproximar les funcions és mitjançant una xarxa neuronal, que permeten aproximar relacions no lineals. Quan s'utilitzen xarxes neuronals profundes (*Deep Learning*) per aproximar funcions de l'RL es parla de la disciplina del ***Deep Reinforcement Learning (deep RL)***.

Malgrat la combinació de les xarxes neuronals amb l'RL fa dècades que s'explora, el *deep RL* ha guanyat importància en els darrers anys amb la combinació de les xarxes neuronals convolucionals amb l'RL, amb algorismes com les Deep Q-Networks (Mnih, et al., 2013) i l'aparició de nous mètodes d'exploració i explotació de les polítiques.

3.2.3 Taxonomia dels mètodes RL

Els mètodes RL es poden classificar de diferents maneres en funció de les múltiples propietats que els caracteritzen. En la bibliografia és habitual veure'ls classificats en funció de tres eixos fonamentals: si utilitzen o no algun tipus de model (*model-based vs. model-free*), si es fonamenten en obtenir primer una estimació de les funcions valor i després la política (*value-based*), o bé si es fonamenten en trobar directament la política òptima en qüestió (*policy-based*). Aquests tres eixos, però, no són excloents, i hi ha metodologies híbrides que exploten el millor de cada eix en diferents moments de l'algorisme RL.

La Figura 11 esquematitza la taxonomia dels mètodes RL i mostra alguns exemples d'algorismes de cada categoria.

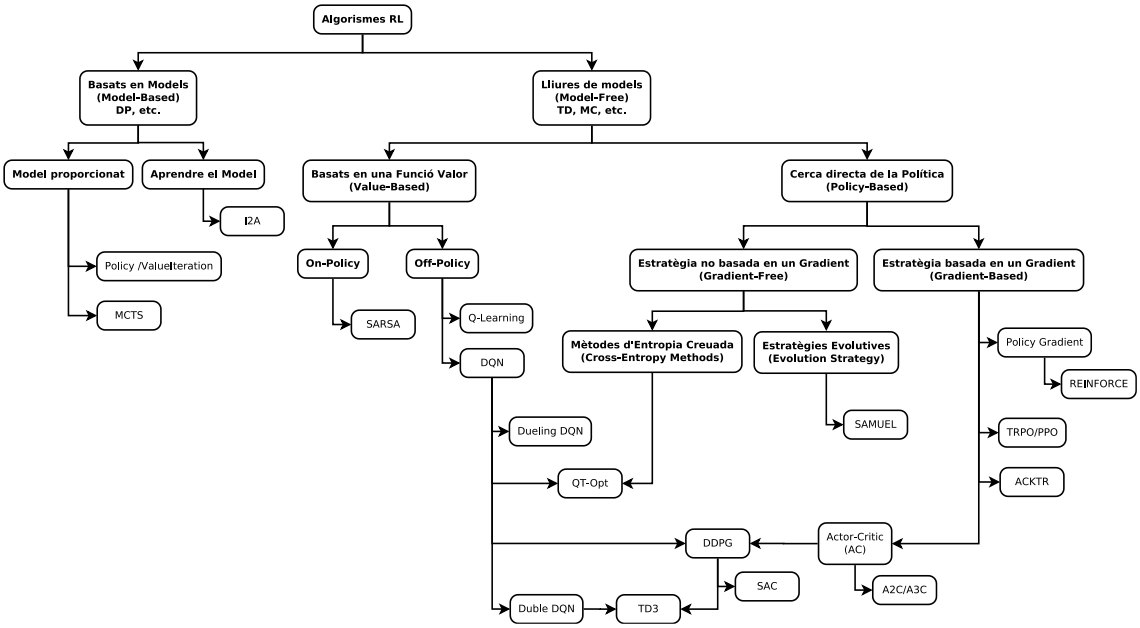


Figura 11. Taxonomia dels mètodes RL. Adaptat de (Zhang & Yu, 2020)

En els següents apartats es descriuran els fonaments dels principals mètodes RL.

3.2.4 Mètodes basats en la Programació Dinàmica (DP)

Aquests mètodes deriven del paradigma del control òptim i parteixen de les següents premisses:

- El sistema és un MDP
- Es té un coneixement complet sobre el model de transició entre estats del sistema ($\mathcal{P}$) així com de la funció d'obtenció de recompenses ($\mathcal{R}$).

En complir-se la propietat de Markov, el problema es pot resoldre de forma recursiva descomponent la funció valor en qüestió en la forma d'una equació de Bellman.

Per exemple, la l'equació de Bellman de la funció valor de l'estat quedaria expressada de la següent manera,

$$v_\pi(s) = \mathbb{E}_\pi[r_t + \gamma R_{t+1}] = \mathbb{E}_\pi[r + \gamma v_\pi(s')] = \sum_{a \in A} \pi(a|s) \sum_{s',r} p(s', r|s, a)\,[r + \gamma v_\pi(s')]$$ Expressió 26

on s' representa qualsevol dels estats als quals es pot arribar en aplicar una acció a a un estat s.

Convé recordar que si aplica la propietat de Markov, el paràmetre t deixa de tenir sentit i per això es reemplaça la nomenclatura s_{t+1} per s'.

Arribat a aquest punt, la forma iterativa per al càlcul del valor de l'estat $v_\pi(s)$ s'expressaria de la manera següent,

$$\begin{cases} v_{\pi,k+1}(s) = \mathbb{E}_\pi\big[r + \gamma v_{\pi,k+1}(s')\big] = \sum_{a \in A} \pi(a|s)[R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)\, v_{\pi,k}(s')] \\ v_{\pi,0}(s) = 0 \end{cases}$$ Expressió 27

Amb aquesta aproximació, quan $k \to \infty$, aleshores $v_{\pi,k}(s) \to v_\pi(s)$.

Els diferents mètodes de programació dinàmica exploten aquestes equacions per resoldre el problema d'optimització de l'agent. En concret, ho fan per a tots els estats i accions possibles, i per aquest motiu reben el nom de mètodes de *full-backup* (en contraposició als mètodes que només utilitzen un subconjunt o mostreig d'aquests estats i accions, anomenats mètodes de *sample-backup*).

Una manera de resoldre aquest problema és mitjançant els mètodes d'Iteració Generalitzada de la Política (*GPI o Generalized Policy Iteration*). Aquests mètodes que combinen iterativament i seqüencial algun mètode per a l'avaluació de la política i, posteriorment, un altre mètode per a l'optimització d'aquesta.

$$\pi_0 \xrightarrow{estimació} v_{\pi_0} \xrightarrow{optimització} \pi_1 \xrightarrow{estim.} v_{\pi_1} \xrightarrow{optim.} \pi_2 \xrightarrow{estim.} \dots \xrightarrow{optim.} \pi^* \xrightarrow{estim.} v_*$$ Expressió 28

A tall d'exemple, es pot utilitzar el mètode d'Iteració sobre la Política (*PI o Policy Iteration*) com a mètode d'estimació de les funcions valor i seguir una selecció *greedy* de les accions d'acord als *V-values* obtinguts per a l'optimització de la política (Figura 10). D'aquesta manera, l'algorisme va convergint a la vegada tant cap a la política òptima com cap als valors òptims (si i només si es té un coneixement complet de les funcions de transició entre estats i de recompensa). Altres mètodes rellevants del paradigma de la programació dinàmica són la Iteració de Valor (*VI o Value Iteration*), les variants asíncrones i les variants aproximades. Per un major aprofundiment en aquestes tècniques veure (Sutton & Barto, Reinforcement Learning, Second Edition : An Introduction, 2018).

Els principals desavantatges de les metodologies basades en la DP són el fet que requereixen tenir un model fiable de les funcions de transició entre estats i de la d'obtenció de la recompensa, així com el seu elevat cost computacional pel fet de treballar amb un *full-backup* (pateixen de la maledicció de la dimensionalitat, *the curse of dimensionality*).

Per lidiar amb aquesta maledicció de la dimensionalitat s'ha combinat aquests mètodes amb aproximacions, per exemple, a les funcions de transició o de recompensa, on aquestes aproximacions poden ser, entre altres, xarxes neuronals entrenades per generalitzar la funció en qüestió. A aquests mètodes se'ls anomena *Approximate Dynamic Programming*.

3.2.5 Mètodes basats en l'algorisme de Monte Carlo (MC)

Els mètodes basats en l'algorisme de Monte Carlo es fonamenten en el **mostreig** aleatori de successos complets que tenen una durada determinada (**episodis**).

En cada succés aleatori, s'escull un estat l'atzar que actuarà com a estat inicial i se li aplica la política en avaluació successivament fins a finalitzar l'episodi, obtenint el valor de recompensa r_t i de retorn a llarg termini R_t per a cada instant de temps t al llarg de l'episodi.

Mitjançant la repetició d'aquest mostreig aleatori un nombre elevat de vegades, calculant els paràmetres de les distribucions empíricament obtingudes, i per la llei dels grans nombres, les estimacions de les funcions valor calculades a partir d'aquests paràmetres estadístics tendeixen cap a les funcions valor de la política en qüestió.

A partir de l'aproximació obtinguda per a la funció valor es pot optimitzar la política en qüestió mitjançant el mètode que es consideri més adient.

El principal avantatge dels mètodes de Monte Carlo és que no requereixen cap tipus de coneixement previ sobre la funció de transició entre estats del sistema ni sobre la funció d'obtenció de la recompensa (es pot provar directament sobre l'entorn). Així mateix, són aplicables també sobre problemes *no Markovians*. Els seus principals desavantatges, però, és que només són vàlids per a tasques episòdiques (és a dir, amb una durada màxima determinada) i que tenen un cost computacional moderat-elevat.

Algunes variacions d'aquests mètodes són la predicció *off-policy* mitjançant el mètode d'*importance sampling,* que permet estimar una política a partir del mostreig de polítiques semblants; la combinació de Monte Carlo amb funcions d'aproximació parametritzades, per exemple per a les funcions valor; el Monte Carlo Tree Search (MCTS), que utilitza MC per generar una planificació prèvia a l'actuació per escollir l'acció potencialment més apropiada.

Per a un millor coneixement d'aquests mètodes, veure (Sutton & Barto, Reinforcement Learning, Second Edition : An Introduction, 2018).

3.2.6 Mètodes basats en la Diferència Temporal (TD)

En contraposició als mètodes estadístics (MC) i probabilístics (DP), els mètodes basats en la Diferència Temporal (TD o *Temporal Difference*) utilitzen *bootstrapping*[12] per al càlcul de les estimacions de les funcions valor.

Aquests mètodes s'apliquen a problemes del tipus MDP i, per tant, s'assumeix que compleixen amb l'expressió

$$v_\pi(s) = \mathbb{E}_\pi[r + \gamma v_\pi(s')] \qquad \text{Expressió 29}$$

de manera que, idealment,

$$\mathbb{E}_\pi[r + \gamma v_\pi(s')] - v_\pi(s) = 0 \qquad \text{Expressió 30}$$

En concret, el *bootstrapping* consisteix en un procediment iteratiu que permet anar actualitzant una estimació de les funcions valor a partir dels valors estimats actuals i la recompensa obtinguda en els estats subseqüents.

A tall d'exemple, es presenta amb més detall aquest procediment per a una estimació $V_\pi(s)$ de la funció valor-estat $v_\pi(s)$.

Inicialment, abans de començar el procediment iteratiu, la funció $V_{\pi,k=0}(s)$ s'inicialitza amb un valor arbitrari, per exemple zero

$$V_{\pi,k=0}(s) = 0 \qquad \text{Expressió 31}$$

En cada iteració k, seguint la política π, l'agent avança cap al següent estat s', obté la recompensa corresponent r_k, i calcula l'anomenat error de la diferència temporal TD mitjançant l'expressió

$$TD = \Big[\big[r_k + \gamma V_{\pi,k-1}(s')\big] - V_{\pi,k-1}(s)\Big] \qquad \text{Expressió 32}$$

Idealment, si $V_{\pi,k}(s) \to v_{\pi,k}(s)$, aleshores $TD \to 0$, per aquest motiu en cada iteració el valor de $V_{\pi,k}(s)$ s'actualitza mitjançant l'expressió

$$V_{\pi,k}(s) \leftarrow V_{\pi,k-1}(s) + \alpha\Big[\big[r_k + \gamma V_{\pi,k-1}(s')\big] - V_{\pi,k-1}(s)\Big] \qquad \text{Expressió 33}$$

on α és l'anomenada **taxa d'aprenentatge (*learning rate*)** i representa la intensitat amb que s'aplica la correcció que representa TD en cada iteració.

D'aquesta manera, a mesura $k \to \infty$, aleshores $V_{\pi,k}(s) \to v_{\pi,k}(s)$.

Els mètodes TD es poden utilitzar tant en l'aprenentatge per episodis (quan es delimita en el temps la durada de la seqüència d'estats) com en l'aprenentatge continu i són els que solen ser computacionalment més eficients.

D'altra banda, existeixen diferents variants del mètode TD que es basen en l'estimació de diferents funcions valor, entre les quals destaquen els mètodes

[12] *Bootstrapping*: procediment pel qual una funció valor s'optimitza (actualitza) a partir dels valors estimats per als estats subseqüents, per exemple: $\boldsymbol{v_{k+1}(s_t) \leftarrow v_k(s_t) + \alpha[r_{t+1} + \gamma v_k(s_{t+1}) - v_k(s_t)]}$

SARSA (*State, Action, Reward, next State, next Action*) i *Q-Learning*. La Taula 7 resumeix els diferents enfocaments d'aquests mètodes.

Mètode	Funció valor estimada i regla d'actualització
TD learning	$V_{\pi,k}(s) \leftarrow V_{\pi,k-1}(s) + \alpha\left[\left[r_k + \gamma V_{\pi,k-1}(s')\right] - V_{\pi,k-1}(s)\right]$
SARSA	$Q_{\pi,k}(s,a) \leftarrow Q_{\pi,k-1}(s) + \alpha\left[\left[r_k + \gamma Q_{\pi,k-1}(s',a')\right] - Q_{\pi,k-1}(s,a)\right]$
Q-Learning	$Q_{\pi,k}(s,a) \leftarrow Q_{\pi,k-1}(s) + \alpha\left[\left[r_k + \gamma \max_{a'} Q_{\pi,k-1}(s',a')\right] - Q_{\pi,k-1}(s,a)\right]$

Taula 7. Regles d'actualització dels principals mètodes TD

A tall d'exemple, a continuació es proporciona el pseudocodi dels algorismes TD learning (per a l'avaluació d'una política π) i Q-Learning (per a l'obtenció de la política òptima).

```
Inicialitzar V de forma arbitrària, p.e. a 0 per a tots els estats
Per cada episodi fer:
  Obtenir l'estat inicial s
  Per cada step t de l'espisoi, si l'estat s no és terminal fer:
    a := escollir acció d'acord a π per a s
    Executar l'acció a i observar la recompensa r i el nou estat s'
    V(s) := V(s) + α[r+γV(s')-V(s)]
    s := s'
  FiPer
FiPer
```

Algorisme 1. TD-Learning per a l'avaluació d'una política, adaptat de (Sutton & Barto, Reinforcement Learning, Second Edition : An Introduction, 2018)

```
Inicialitzar Q arbitràriament, p.e. a 0 per a tot parell estat-acció
Per cada episodi fer:
  Obtenir l'estat inicial s
  Per cada step t de l'espisoi, si l'estat s no és terminal fer:
    a := seleccionar acció via ε-greedy respecte a arg max_a Q(s,a)
    Executar l'acció a i observar la recompensa r i el nou estat s'
    Q(s,a) := Q(s,a) + α[r+γ max_a' Q(s',a') -Q(s,a)]
    s := s'
  FiPer
FiPer
```

Algorisme 2. Q-Learning per a l'estimació on-policy de Q, adaptat de (Sutton & Barto, Reinforcement Learning, Second Edition : An Introduction, 2018)

3.2.6.1 Multi-step bootstrapping

Les regles d'actualització de la funció valor que presentades fins al moment només tenen en compte l'error que es genera en una diferència temporal d'una única unitat de temps (*time step*), però existeixen variants que tenen en compte un nombre més elevat d'unitats. En el límit, si es treballa amb episodis i es té en compte la durada de tot l'episodi, aquest algorisme esdevé l'algorisme de Monte Carlo (MC).

Per indicar el nombre d'unitats que es tenen en compte s'utilitza el paràmetre $\lambda \in \mathbb{Z}^+$, i els algorismes reben el nom de TD(λ), SARSA(λ) o Q(λ), respectivament.

A tall d'exemple, en el cas de TD(λ) el càlcul de l'error en la diferència temporal es dura a terme mitjançant l'expressió

$$TD(\lambda = n) = \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^n V_{\pi,k-1}(s_{t+n})\right] - V_{\pi,k-1}(s_t)$$

Expressió 34

El cas trivial en què $\lambda = 0$ es correspon al mètode TD original.

3.2.6.2 Double Q-Learning

Un problema que acostumen a tenir els algorismes de Q-Learning és que en utilitzar els mateixos valors de Q per seleccionar l'acció potencialment millor així com per estimar la nova Q, això fa que sigui més fàcil seleccionar accions amb valors de Q sobreestimats i més difícil seleccionar accions amb valors de Q subestimats.

Per intentar corregir aquest fet es pot emprar estratègies com el Double QL (també en versions triple, quàdruple, etc.). En el cas del Double QL es mantenen dues instàncies de Q (per exemple Q i Q'), de manera que en cada time step s'actualitza aleatòriament una de les dues amb el valor que s'obté de l'altra.

$$aleatòriament \begin{cases} Q_k(s,a) \leftarrow Q'_{k-1}(s) + \alpha\left[\left[r_k + \gamma \max_{a'} Q'_{k-1}(s',a')\right] - Q'_{k-1}(s,a)\right] \\ Q'_k(s,a) \leftarrow Q_{k-1}(s) + \alpha\left[\left[r_k + \gamma \max_{a'} Q_{k-1}(s',a')\right] - Q_{k-1}(s,a)\right] \end{cases}$$

Expressió 35

Per a la selecció de l'acció, en canvi, es pot utilitzar la mitjana de les dues Q, combinat amb un $\in$-*greedy*.

Aquest mètode permet obtenir millors resultats que l'algorisme QL original en força escenaris, per exemple, en la seva variant Double DQN, aplicada en els jocs de la màquina ATARI (van Hasselt, Guez, & Silver, Deep Reinforcement Learning with Double Q-learning, 2015).

3.2.6.3 Aproximacions amb xarxes neuronals: les DQN i altres estratègies

Els algorismes TD descrits fins al moment tenen l'inconvenient que cal anar emmagatzemant de forma tabular els resultats de les funcions valor per a la seva posterior explotació. Evidentment, si el nombre d'estats o accions és elevat (o si és continu), això és inviable. En aquests casos cal dur a terme algun tipus d'aproximació a aquestes funcions.

Tot i que s'han provat diferents tipus d'aproximacions a les funcions valor en combinació amb els mètodes TD, la que ha tingut més èxit en els darrers anys ha estat l'ús de xarxes neuronals per a aquest propòsit.

El primer treball disruptiu en aquest sentit va ser el desenvolupament de la *Deep Q-Network* (Mnih, et al., 2013) (Mnih, et al., 2015), que va establir una arquitectura *end-to-end* capaç d'aconseguir uns nivells de joc similars o superiors als dels humans per una gran quantitat de jocs de la consola Atari tenint com a única entrada de l'agent els *frames* de la pantalla del joc, un valor equivalent a la puntuació (normalitzat per als diferents jocs) i, si s'escau, el nombre de vides restants.

La DQN es basa en el mètode *Q-Learning* i aproxima a la vegada, mitjançant una xarxa neuronal convolucional, la funció valor-acció Q_a per a cada acció possible, així com la funció valor-acció òptim (Q_*). En l'article original, l'entrada a la xarxa són 4 *frames* consecutius (la qual cosa li permet estimar les trajectòries) i la puntuació obtinguda com a resultat d'executar l'acció anterior (el *reward*). La xarxa té tantes sortides com accions es poden executar en el joc (espai d'accions discret i molt reduït) i cada sortida representa el valor-acció de l'acció que té associada (Q_a). El valor més elevat de Q_a es correspon al valor de Q_* i, per tant, a l'acció òptima, que serà la que l'agent escollirà i executarà durant els 4 *frames* següents.

Un element important de la DQN és la seva estratègia d'*experience replay* per evitar l'*overfitting* respecte a la política en explotació/exploració, tot disminuint les correlacions temporals entre estats. Aquesta estratègia es basa en l'ús d'una memòria (*replay buffer*) en la qual s'emmagatzemen les experiències viscudes per l'agent com a tuples de la forma $(s_t, a_t, r_{t+1}, s_{t+1})$. El seu funcionament s'explica a continuació.

La DQN s'executa a episodis de durada màxima T. En cada instant t d'un episodi se seleccionen J experiències prèvies a l'atzar del *buffer replay* amb una probabilitat uniforme. Cadascuna d'aquestes experiències es correspon a una tupla amb la forma (s_j, a_j, r_j', s_j'), on el subíndex j referencia cada experiència concreta dins el conjunt de J experiències prèvies[13].

Per estabilitzar l'algorisme, la DQN divideix el seu entrenament en diferents períodes de durada C i utilitza en cadascun d'ells una xarxa neuronal auxiliar $Q_*^-(s, a; \theta^-)$ que es manté inalterada durant tot el període.

Aquesta xarxa Q_*^- es correspon a l'estat de la xarxa en estimació Q_* a l'inici del període. La finalitat d'aquesta xarxa Q_*^- és la de calcular el valor-acció òptim estimat per l'algorisme d'acord a l'aprenentatge efectuat fins al darrer període finalitzat. Amb aquests valors es fixaran els objectius per a l'actualització de la xarxa Q_* (la que està en aprenentatge).

D'aquesta manera, en cada instant t, la xarxa Q_* s'actualitza tot fixant un valor esperat y_j (*target*) per a cada experiència prèvia j d'acord amb la següent expressió:

$$y_j = r_j' + \gamma \max_{a'} Q_*^-(s_j', a_j'; \theta^-) \qquad \text{Expressió 36}$$

Amb el cas trivial

$$y_j = r_j' \qquad \text{Expressió 37}$$

si s_j' es corresponia a un estat terminal o de fi d'episodi.

[13] En complir-se la propietat de Markov, s'elimina la referència temporal t+1 i es reemplaça pel símbol ', de manera que s' = s_{t+1}

Amb aquests valors esperats (*target*), la xarxa auxiliar Q_* s'entrena mitjançant un descens de gradient mitjançant el mètode que es desitgi (SGD, RMSprop, Adam, etc.) respecte al paràmetre θ d'acord a la següent expressió

$$\left(y_j - Q_*(s_j, a_j; \theta)\right)^2 \qquad \text{Expressió 38}$$

Atès que la xarxa Q_*^- és la que s'utilitza per al càlcul dels valors objectiu (*target*) per a la xarxa Q_*, aquesta xarxa Q_*^- rep el nom de xarxa per als objectius (***target network***)[14].

En finalitzar cada període d'entrenament, la xarxa *target* pren els paràmetres (pesos) de la xarxa neuronal en entrenament (Expressió 39), o bé s'actualitza a partir d'aquesta amb un factor d'actualització *tau* (τ) (Expressió 40).

$$\theta^- = \theta \qquad \text{Expressió 39}$$

$$\theta^- = \tau \cdot \theta + (1 - \tau) \cdot \theta^- \qquad \text{Expressió 40}$$

A continuació, el procediment comença de nou per al següent període.

Al llarg de l'episodi, s'utilitza una **política d'exploració** per anar introduint experiències al *buffer replay*. Aquesta política es una política ϵ-*greedy* aplicada sobre la **política d'explotació**. Això vol dir que, per tal d'explorar noves possibilitats, amb una probabilitat ϵ se seleccionarà una acció a l'atzar, independentment de la política d'explotació, mentre que amb una probabilitat 1-ϵ l'acció seleccionada serà determinada per la política d'explotació actual. Malgrat aquesta política d'exploració basada en la política d'explotació, l'aprenentatge es du a terme a partir del *buffer replay* i, per tant, es considera un **mètode *off-policy***.

Tots aquests elements han estat claus per a l'èxit de les DQN. A continuació es mostra l'estructura de l'algorisme per a una major comprensió.

```
Inicialitzar la replay memory D
Inicialitzar el paràmetre θ de la funció Q_θ amb valors
aleatoris
Inicialitzar el paràmetre θ⁻ de la funció diana Q_θ- amb
θ⁻= θ
Per cada episodi fer:
  Obtenir l'estat inicial s1
  Per cada step t de l'espisoi, si l'estat st no és
terminal fer:
    #Exploració:
    at := seleccionar acció via ε-greedy respecte a
arg max_a Q_θ(s_t, a)
    Executar l'acció at i observar rt+1 i st+1
```

[14] El nom *target network* pot resultar confús, ja que pot fer pensar que es tracta de la xarxa objectiu, aquella que es vol obtenir, i no és així, és la que s'utilitza per calcular els objectius (*targets*) de l'aprenentatge

```
    Desar la transició (st,at,rt+1,st+1) en la memoria D
    #Experience replay:
    Seleccionar a l'atzar N transicions de la memoria D
    Per cada transició Tj:=(st',at',rt'+1,st'+1) fer:
      Si l'episodi st'+1 és terminal llavors:
        yj := rt'+1
      Sinó:
        yj := rt'+1 + γ max_a Q_θ⁻(s_t'+1, a)
      FiSi
      #Estimació de Q:
      Fer un descens de gradient per (y_j − Q_θ⁻(s_t', a_t'))² respecte a θ
    Cada C time steps, θ⁻= θ
  FiPer
FiPer
```

Algorisme 3. DQN per a l'avaluació d'una política, adaptat de (Mnih, et al., 2015)

Posteriorment a la formulació del DQN han aparegut modificacions i millores d'aquest, com són les estratègies de *prioritized experience replay*, o els algorismes de *dueling DQN* (que utilitzen la funció avantatge per estimar simultàniament la funció valor-estat V i la funció avantatge A i combinar-les posteriorment Q=V+A, ressaltant així molt més les diferències entre accions amb valors-acció propers) i *double DQN* (una adaptació de l'algorisme Double QL, vist anteriorment, al paradigma DQN) entre altres. Per a més informació, veure (Sutton & Barto, Reinforcement Learning, Second Edition : An Introduction, 2018).

També convé destacar que l'algorisme DQN i les seves variants només es vàlid en **espais d'accions discrets finits**. Per al cas d'espais d'accions continus, una

Algorithm 1 Continuous Q-Learning with NAF

Randomly initialize normalized Q network $Q(\boldsymbol{x}, \boldsymbol{u}|\theta^Q)$.
Initialize target network Q' with weight $\theta^{Q'} \leftarrow \theta^Q$.
Initialize replay buffer $R \leftarrow \emptyset$.
for episode=1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state $\boldsymbol{x}_1 \sim p(\boldsymbol{x}_1)$
  for t=1, T **do**
    Select action $\boldsymbol{u}_t = \mu(\boldsymbol{x}_t|\theta^\mu) + \mathcal{N}_t$
    Execute $\boldsymbol{u}_t$ and observe r_t and $\boldsymbol{x}_{t+1}$
    Store transition $(\boldsymbol{x}_t, \boldsymbol{u}_t, r_t, \boldsymbol{x}_{t+1})$ in R
    for iteration=1, I **do**
      Sample a random minibatch of m transitions from R
      Set $y_i = r_i + \gamma V'(\boldsymbol{x}_{i+1}|\theta^{Q'})$
      Update θ^Q by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(\boldsymbol{x}_i, \boldsymbol{u}_i|\theta^Q))^2$
      Update the target network: $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
    end for
  end for
end for

Algorisme 4. Continuous Q-Learning mitjançant NAF, extret de (Gu, Lillicrap, Sutskever, & Levine, Continuous Deep Q-Learning with Model-based Acceleration, 2016)

alternativa són les **Normalized Advantage Functions (NAF)** (Gu, Lillicrap, Sutskever, & Levine, 2016), un algorisme que permet treballar de forma anàloga a les DQN en problemes amb un espai continu d'accions.

3.2.7 Mètodes basats en l'optimització directa de la política (*Policy-based Methods*)

A diferència dels mètodes basats en les funcions valor, els ***mètodes basats en la política*** cerquen directament la política òptima sense necessitat de mantenir una funció valor a tal efecte.

Aquests mètodes aproximen la política òptima mitjançant una funció parametritzada (amb paràmetre θ), típicament una xarxa neuronal (si bé es pot tractar també d'altres tipus de funcions).

$$\pi_\theta \approx \pi^* \qquad \text{Expressió 41}$$

El problema queda definit doncs de la següent manera

$$\theta = \arg\max_\theta \mathbb{E}[R_t|\pi_\theta] \qquad \text{Expressió 42}$$

Per tant, en aquest cas, el que serà necessari és o bé provar diferents combinacions dels valors de θ fins arribar a una política òptima (com fan, per exemple, els algorismes genètics), o be fer un ascens pel gradient de la funció d'aproximació en relació al paràmetre θ (com fan els algorismes basats en gradient).

El principal avantatge d'aquests mètodes és que permeten treballar en espais continus d'accions o d'elevada dimensionalitat, tenen una millor convergència i poden aprendre polítiques estocàstiques. El principal desavantatge és que si no es vigila en el seu disseny, poden tendir a màxims locals i, per tant, polítiques subòptimes.

3.2.7.1 Mètodes basats en gradient (gradient-based)

Els mètodes basats en gradient aproximen directament la política mitjançant una funció diferenciable respecte a θ, tot ajudant-se d'una funció de mesura de l'eficiència de la política $J(\theta_t)$ a partir de la qual es calcula un gradient $\nabla J(\theta_t)$ que permet anar guiant l'exploració, de forma iterativa, cap a les regions més prometedores de l'espai de polítiques (aquelles que indica el gradient).

$$\nabla_{\theta_t} J(\theta_t) = \begin{pmatrix} \frac{\partial J(\theta_t)}{\partial \theta_{1,t}} \\ \vdots \\ \frac{\partial J(\theta_t)}{\partial \theta_{n,t}} \end{pmatrix} \qquad \text{Expressió 43}$$

L'actualització del paràmetre θ es pot dur a terme, per exemple, de la següent manera

$$\theta_{t+1} \leftarrow \theta_t + \alpha \nabla J(\theta_t) \qquad \text{Expressió 44}$$

On $\alpha \in \mathbb{R}^+$ és la taxa d'aprenentatge en cada iteració.

El mètode REINFORCE (Williams, 1992), per exemple, va generant episodis complets (en la línia de Monte Carlo) i, per a cada instant de cada episodi, calcula utilitza R_t i utilitza la següent regla d'actualització.

$$\theta_{t+1} \leftarrow \theta_t + \alpha\gamma^t \mathrm{R}_t \nabla \ln \pi(a_t|s_t,\theta) \qquad \text{Expressió 45}$$

Per dur a terme l'ascens en el gradient es pot emprar una estratègia d'ascens estocàstic (***Stochastic Gradient Ascent***), utilitzant el mètode de Monte Carlo per computar el gradient.

Existeixen múltiples variants dels mètodes basats en gradients:

- El mètode ***REINFORCE with Baseline*** resta a l'R_t una línia de base per intentar disminuir la variància en el gradient. Hi ha diferents variants respecte a aquest mètode en funció de la línia de base que s'utilitzi. Una línia de base natural pot ser, per exemple, la mitjana de les recompenses.

- El mètode ***Trust Region Policy Optimization (TRPO)*** utilitza la mesura Kullbeck-Leibler de divergència entre distribucions per comparar la política potencialment òptima actual i la nova política en exploració, amb l'objectiu de garantir que la nova política en exploració es troba dins uns marges de confiança dins l'espai de polítiques en exploració (Schulman, Levine, Moritz, Jordan, & Abbeel, 2017).
- El mètode ***Proximal Policy Optimization (PPO)*** simplifica la implementació de TRPO i el millora considerablement (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017).

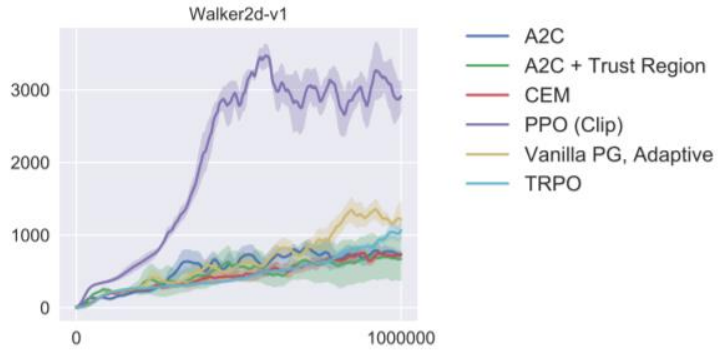


Figura 12. Rendiment de PPO per a l'entorn Walker2D d'OpenAI amb l'entorn físic MuJoCo comparat amb altres algorismes d'espai d'accions continu, extret de (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017)

3.2.7.2 Mètodes off-policy basats en gradient (gradient-based)

En els darrers anys també s'han desenvolupat mètodes basats en gradients i off-policy, entre aquests destaquen el **Soft Q-Learning** (Haarnoja, Tang, Abbeel, & Levine, 2017), que utilitza una aproximació basada en l'energia i entropia generades per la política, i el mètode **Path Consistency Learning** (Nachum, Norouzi, Xu, & Schuurmans, 2017).

3.2.7.3 Mètodes sense gradient (gradient-free)

Els mètodes sense gradient utilitzen altres tipus d'aproximacions que generar els valors de θ, com són els algorismes evolutius i els d'entropia creuada.

Dins els algorismes evolutius, els algorismes genètics, per exemple, proven diferents combinacions de valors per a θ, on cada configuració de θ es pot considerar un individu, i cada element de θ, un gen. Mitjançant la generació aleatòria d'individus s'obtindria una població a la qual s'aplicarien procediments de mutació i de recombinació, tot comprovant l'eficàcia dels diferents individus en l'entorn i seleccionant aquells que donen millors resultats de generació en generació. Es diuen algorismes evolutius perquè simulen l'evolució biològica de les poblacions. Entre aquests destaca recentment **Evolution Strategies (ES)** (Salimans, Ho, Chen, Sidor, & Sutskever, 2017).

Els mètodes d'entropia creuada (CEM) es basen en diferència relativa d'entropia entre les polítiques actuals i aquella cap a la que convergeixen al llarg del procés d'exploració (idealment, l'òptima). En aquests mètodes s'escullen a l'atzar N mostres de θ, de les quals se seleccionen les M millors a partir de les quals es parametritza una distribució gaussiana. Iterativament es repeteix el procés amb N mostres més d'aquesta gaussiana obtinguda, fins que el resultat convergeix. Entre aquests mètodes destaca QT-Opt (Kalashnikov, et al., 2018).

3.2.8 Mètodes Actor-Crític (AC)

Aquests mètodes permeten l'aprenentatge simultani d'una política i d'una funció valor, de manera que la funció valor en aprenentatge guia el camí que segueix l'exploració de la política.

Així doncs, en els mètodes AC, una part de l'algorisme, l'anomenada **crític**, s'encarrega d'aprendre aquesta funció valor, mentre que una altra part de l'algorisme, l'anomenada **actor**, s'encarrega d'explorar l'espai de polítiques de forma directa amb l'ajuda de la funció valor estimada pel crític.

L'Algorisme 5 mostra un exemple d'Actor Crític en què s'utilitza la funció avantatge com a element que guia l'obtenció de l'aproximació tant a la funció valor com a la política.

```
Inicialitzar el paràmetre θ de la política π_θ amb valors aleatoris
Inicialitzar el paràmetre w de la funció valor v_w amb valors aleatoris
Per cada episodi fer:
  Obtenir l'estat inicial s_1
  Per cada step t de l'espisoi, si l'estat s_t no és terminal fer:
    a_t := seleccionar acció d'acord a π_θ(s_t)
    Executar l'acció a_t i observar r_{t+1} i s_{t+1}
    #Calcular l'avantatge de l'acció escollida d'acord a v_w:
    δ_t = r_{t+1} + γ v_w(s_{t+1}) − v_w(s_t)
    #Actualitzar la funció valor (crític):
    w ← w + β δ_t ∇_w v_w(s_t)
    #Actualitzar la política (actor):
    θ ← θ + α δ_t ∇_θ log π_θ(a_t|s_t)
  FiPer
FiPer
```

Algorisme 5. Exemple d'Actor Crític basat en la funció avantatge (Advantage Actor Critic), adaptat de (van Hasselt, 2018)

Hi ha múltiples variants d'aquest mètode, en funció de l'algorisme que s'utilitzi per a l'estimació de la funció valor i del que s'utilitzi per a l'exploració de la política.

Entre els més destacats es troben el ***Deterministic Policy Gradient o DPG*** (Silver, et al., 2014) i el ***Deep DPG*** (Lillicrap, et al., 2015), tots dos capaços de treballar en espais d'accions continus, el darrer amb resultats molt competents; i les variants asíncrones ***Asynchronous Advantage Actor Crític o A3C*** (Mnih, et al., 2016), que permeten paral·lelitzar el procés.

Així mateix, també destaca el mètode ***Soft Actor Critic (SAC)*** (Haarnoja, Zhou, Abbeel, & Levine, 2018), que a diferència dels anteriors, es tracta d'un mètode ***off-policy***.

3.2.9 Mètodes Basats en Models

Finalment, els diferents algorismes d'RL poden fer ús de models per ajudar-se en la seva tasca. Un model és una representació formal d'algun aspecte de l'entorn o dels coneixements previs i/o adquirits de l'agent, entre altres.

Sovint, en l'RL, aquests models s'utilitzen per predir com es comportarà l'entorn, de manera que l'agent pot explorar aquesta simulació de l'entorn sense executar les accions directament en aquest, de manera que pugui predir el resultat de seguir determinada política o executar determinades accions. A aquesta fase de predicció mitjançant el model se l'anomena fase de **planificació**.

També alguns algorismes utilitzen aquests models de l'entorn per aprendre les funcions valor mitjançant una exploració simulada, a la qual poden donar un pes similar o no a l'exploració de l'entorn, però els permet anticipar-se en la **planificació** amb el coneixement previ que ha anat adquirint de l'entorn al llarg de la seva experiència.

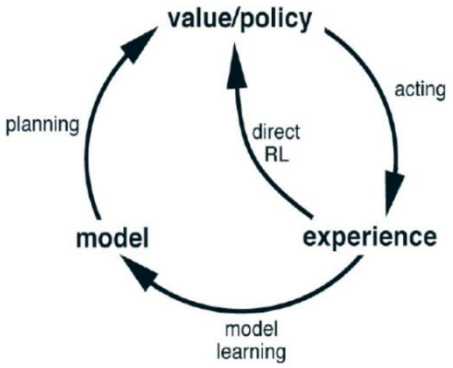


Figura 13. Model-based RL, imatge extreta de (Sutton & Barto,

Tots aquests mètodes que utilitzen models se'ls anomena ***model-based***, en contraposició als mètodes que no utilitzen cap model explícit durant el seu funcionament (mètodes ***model-free***).

Els models, a la vegada, són funcions, i poden ser de diferents tipus (tabulars, funcions lineals, distribucions com poden ser gaussianes, xarxes neuronals, etc.).

Els mètodes *model-based* poden aprendre el model a partir de l'experiència o bé se'ls pot facilitar com a coneixement previ a la seva execució. A la seva vegada, aquests models poden consistir en funcions d'aproximació parametritzades.

Finalment, també hi ha treballs que utilitzen **mètodes mixtes**, com ara els que utilitzen un model per fixar una política de base considerada més o menys segura o eficient però que cal optimitzar (per exemple perquè el model és imperfecte).

Entre els mètodes actuals basats en models destaquen els següents:

- L'algorisme ***Dyna-Q*** (Sutton, 1990), permet aprendre un model a partir de l'experiència i explotar-lo durant la fase de planificació per anar aprenent la funció valor i/o la política que corresponguin. L'algorisme considera l'experiència real i la simulada com a equivalents.

```
Inicialitzar Q(s,a) i Model(s,a) per a tot s∈S i tota a∈A

Per sempre fer:
  s ← obtenir l'estat actual
  a ← seleccionar acció via ∈-greedy a partir d's i Q
  Executar l'acció a sobre l'entorn real i observar r i s'
  #Actualitzar la funció valor amb l'experiència real:
  Q(s,a) ← Q(s,a) + α[r + γ max_a' Q(s',a') − Q(s,a)]
  #Actualitzar el model:
  Model(s,a)← s',r
  #Planificació:
  Repetir N cops:
    s ← seleccionar aleatòriament un estat prèviament observat
    a ← seleccionar aleatòriament una acció prèviament aplicada a s
    # Executar l'acció a sobre l'entorn simulat i observar r i s'
    s',r ← Model(s,a)
    #Actualitzar la funció valor amb l'experiència simulada:
    Q(s,a) ← Q(s,a) + α[r + γ max_a' Q(s',a') − Q(s,a)]
```

Algorisme 6. Dyna-Q, adaptat de (Sutton & Barto, 2018)

- També existeixen variants del mètode Dyna-Q per als espais d'accions continus, com l'algorisme proposat per (Gu, Lillicrap, Sutskever, & Levine, 2016), que combina NAF amb una variant de Dyna-Q.

Algorithm 2 Imagination Rollouts with Fitted Dynamics and Optional iLQG Exploration

Randomly initialize normalized Q network $Q(\boldsymbol{x}, \boldsymbol{u}|\theta^Q)$.
Initialize target network Q' with weight $\theta^{Q'} \leftarrow \theta^Q$.
Initialize replay buffer $R \leftarrow \emptyset$ and fictional buffer $R_f \leftarrow \emptyset$.
Initialize additional buffers $B \leftarrow \emptyset, B_{old} \leftarrow \emptyset$ with size nT.
Initialize fitted dynamics model $\mathcal{M} \leftarrow \emptyset$.
for $episode = 1, M$ **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state $\boldsymbol{x}_1$
  Select $\mu'(\boldsymbol{x}, t)$ from $\{\mu(\boldsymbol{x}|\theta^\mu), \pi_t^{iLQG}(\boldsymbol{u}_t|\boldsymbol{x}_t)\}$ with probabilities $\{p, 1-p\}$
  for $t = 1, T$ **do**
    Select action $\boldsymbol{u}_t = \mu'(\boldsymbol{x}_t, t) + \mathcal{N}_t$
    Execute $\boldsymbol{u}_t$ and observe r_t and $\boldsymbol{x}_{t+1}$
    Store transition $(\boldsymbol{x}_t, \boldsymbol{u}_t, r_t, \boldsymbol{x}_{t+1}, t)$ in R and B
    if $\mod(episode \cdot T + t, m) = 0$ and $\mathcal{M} \neq \emptyset$ **then**
      Sample m $(\boldsymbol{x}_i, \boldsymbol{u}_i, r_i, \boldsymbol{x}_{i+1}, i)$ from B_{old}
      Use $\mathcal{M}$ to simulate l steps from each sample
      Store all fictional transitions in R_f
    end if
    Sample a random minibatch of m transitions $I \cdot l$ times from R_f and I times from R, and update $\theta^Q, \theta^{Q'}$ as in Algorithm 1 per minibatch.
  end for
  if B_f is full **then**
    $\mathcal{M} \leftarrow$ FitLocalLinearDynamics(B_f) (see Section 5.3)
    $\pi^{iLQG} \leftarrow$ iLQG_OneStep($B_f, \mathcal{M}$) (see appendix)
    $B_{old} \leftarrow B_f, B_f \leftarrow \emptyset$
  end if
end for

Algorisme 7. Variant de Dyna-Q combinada amb NAF, extret de (Gu, Lillicrap, Sutskever, & Levine, Continuous Deep Q-Learning with Model-based Acceleration, 2016)

- Finalment, destacar els mètodes de planificació basats en simulació dels possibles futurs per mostreig de Monte Carlo, com són la cerca simple (Simple MC Search) i la cerca en arbre (Monte Carlo Tree Search, o MCTS).

3.3 Altres enfocaments interessants per al problema del TFM en qüestió

Safe Reinforcement Learning (García & Shafie, 2020)

Es tracta d'un enfocament pensat per minimitzar el risc de caiguda quan es trasllada l'algorisme a un robot real. El mètode combina la metodologia *model-based* amb la metodologia *model-free*, de manera que es proporciona una política de base que es considera segura (per exemple, obtinguda en un simulador), i s'aplica RL per optimitzar-la en el món real. A diferència d'altres mètodes, però, en el moment de l'exploració, en comptes d'utilitzar algorismes com €-*greedy*, el mètode fa una estimació del risc de caiguda i, si preveu que és molt elevat, aleshores opta per escollir una acció de la política de base, amb la intenció així de minimitzar-lo.

Paired Open-Ended Trailblazer (Wang, Lehman, Clune, & Stanley, 2019)

Aquest mètode utilitzar mètodes evolutius per generar un currículum de dificultat incremental, en principi sense límit en la seva evolució (*open-ended*), de manera

que l'agent va evolucionant progressivament en cadascun d'aquests entorns. El terme *paired* fa referència a la coevolució de l'agent i l'entorn, de manera que els salts en la dificultat de l'entorn venen determinats pel grau d'èxit de l'agent en els entorns que ha anat explorant. En l'experiment, els autors fan servir algorismes evolutius de tipus *Evolution Strategies* o *ES*[15], malgrat indiquen que es podria fer servir qualsevol altre mètode d'interès, com els actor-crític o els basats en gradients. El més rellevant de l'estudi és que mitjançant la coevolució aconsegueixen comportaments que no apareixen quan se segueix un currículum prefixat o quan s'entrenen els agents només amb l'entorn que s'està provant.

Reinforcement Learning for Improving Agent Design (Ha, 2018)

Curiós article en què l'objectiu del RL és no només optimitzar la política sinó també la forma del propi robot per tal d'incrementar el seu èxit en l'entorn[16].

CAQL: Continuous Action Q-Learning (Ryu, Chow, Anderson, Tjandraatmadja, & Boutilier, 2019)

Es tracta d'un framework d'algorismes molt punters que permeten abordar el problema de l'RL en espais continus d'accions des de la perspectiva del Q-learning millorant els algorismes i resultats obtinguts fins al moment (com les funcions NAF). Els autors desenvolupen un algorisme de base i diversos elements *plug-and-play*, de manera que l'algorisme es pot adaptar a diferents problemes.

[15] Hi ha un article interessant per tots els recursos gràfics i exemples que ofereix sobre les ES en el següent enllaç https://blog.otoro.net/2017/10/29/visual-evolution-strategies/

[16] Per a més informació (inclosos vídeos) veure: https://designrl.github.io/

4 La Plataforma OpenAI Gym

4.1 Elements de l'RL en OpenAI Gym

Per explicar els elements de l'RL en l'OpenAI Gym es partirà del següent exemple:

```
import gym

N_EPISODES = 10
MAX_T_EPISODE = 50

class Policy:
    def getAction(self, state):
        return env.action_space.sample()

policy = Policy()

env = gym.make('BipedalWalker-v3')

for i_episode in range(N_EPISODES):
    observation = env.reset()
    env.render()
    for t in range(MAX_T_EPISODE):
        observation, reward, done, info =
            env.step(policy.getAction(observation))
        env.render()
        if done:
            print("Episode finished after {} timesteps".format(t+1))
            break
env.close()
```

4.1.1 L'entorn

Com es pot observar, la funció **gym.make** permet instanciar un entorn predefinit tot passant-li com a paràmetre el nom d'aquest entorn. La instanciació requereix indicar també, juntament amb el nom, la versió de l'entorn per tal de garantir la reproductibilitat dels experiments.

Per a la versió actual d'OpenAI Gym, els entorns BipedalWalker disponibles són el *BipedalWalker-v3* i el *BipedalWalkerHardcore-v3*.

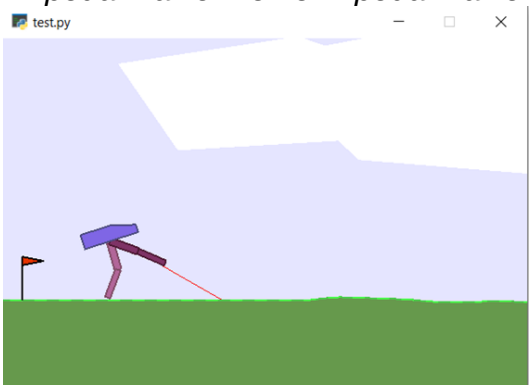


Figura 14. Entorn BipedalWalker-v3

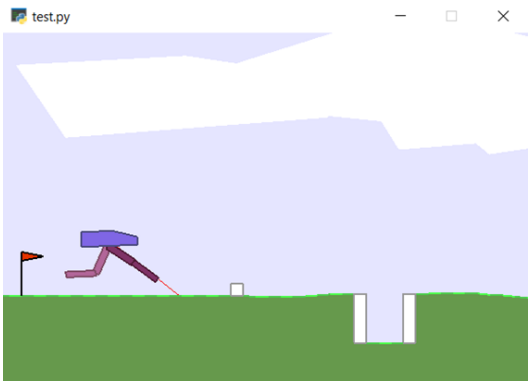


Figura 15. Entorn BipedalWalkerHardcore-v3

L'entorn BipedalWalker-v3 consta d'un terreny sense obstacles, amb lleus desnivells (Figura 14). L'entorn BipedalWalkerHardcore-v3, en canvi, conté

diferents tipus d'obstacles, com són escales (*stairs*), pous (*pits*), petits murs o ressalts (stumps), a part dels desnivells del terreny.

En aquests entorns, la simulació física es du a terme mitjançant la biblioteca Box2D.

D'aquesta manera, l'entorn obtingut serà una instància d'una classe concreta derivada de la ***gym.Env***. Aquest objecte disposarà, entre altres, dels mètodes següents:

- **reset**: funció que, d'una banda reinicia l'entorn (iniciant així un nou episodi), i de l'altra retorna una observació d'aquest estat inicial.
- **step(action)**: funció que executa l'acció passada com a paràmetre en l'entorn i retorna una tupla amb la següent informació:
    - Una observació de l'estat al qual s'ha transicionat en haver executat l'acció
    - La recompensa rebuda com a conseqüència de l'acció
    - Un booleà que indica si el nou estat de l'entorn és un estat terminal o no
    - Informació addicional d'interès, segons l'entorn
- **render**: permet dibuixar l'entorn actual
- **close**: permet alliberar els recursos assignats a l'entorn

4.1.2 L'espai d'observacions (estats)

Les observacions, internament, consten d'arrays n-dimensionals de NumPy. El nombre de components de l'array i els valors que poden prendre conforma l'espai d'observacions.

Hi ha diferents tipus d'espais predefinits: Box, Discrete, Dict, MultiBinary, MultiDiscrete i Tuple.

Per als entorns BipedalWalker, l'espai d'observacions es troba definit com una Box(24,), és a dir, una hipercaixa de 24 dimensions. Així doncs, cadascuna de les observacions tindrà 24 components, cadascuna d'elles amb valor del domini dels nombres reals, en aquest cas, no acotat.

Es pot consultar la definició de l'espai mitjançant:

```
>>> print(env.observation_space)
Box(24,)
```

D'altra banda, les cotes es poden consultar mitjançant:

```
>>> print(env.observation_space.low)
[-inf -inf -inf -inf -inf -inf -inf -inf -inf -inf -inf -inf -inf -inf
-inf -inf -inf -inf -inf -inf -inf -inf -inf -inf]

>>> print(env.observation_space.high)
[inf inf inf inf inf inf inf inf inf inf inf inf inf inf inf inf inf inf
inf inf inf inf inf inf]
```

D'acord a la documentació disponible en el codi font de l'entorn[17], aquestes 24 components es corresponen a les següents mesures, en aquest ordre:

- Angle d'inclinació del cos (hull angle)
- Velocitat angular del cos (hul angular velocity)
- Velocitat horitzontal (en l'eix de les x)
- Velocitat vertical (hip angle)en l'eix de les y)
- Angle del maluc de la primera cama (hip angle)
- Velocitat angular del maluc de la primera cama (hip angular speed)
- Angle del genoll de la primera cama (knee angle)
- Velocitat angular del genoll de la primera cama (knee angular speed)
- Contacte del peu de la primera cama amb el terra (ground contact)
- Angle del maluc de la segona cama (hip angle)
- Velocitat angular del maluc de la segona cama (hip angular speed)
- Angle del maluc de la segona cama (knee angle)
- Velocitat angular del maluc de la segona cama (knee angular speed)
- Contacte del peu de la segona cama amb el terra (ground contact)
- 10 mesures LIDAR (*Light Detection and Ranging*), que en el món real es correspondria a una tecnologia òptica de teledetecció que permetria mesurar la distància des de l'emissor LIDAR, fins topar amb qualsevol objecte o superfície que en reflexi la llum.

Com es pot observar, no hi ha cap mesura de coordenades en les observacions de l'entorn.

4.1.3 L'espai d'accions

Les tipologies dels espais d'accions es defineixen de la mateixa manera que les dels espais d'observacions.

En el cas dels BipedalWalkers, aquest espai d'accions està definit com una Box(4,) acotada a [-1.0,1.0] en totes les seves dimensions.

```
>>> print(env.action_space)
Box(24,)

>>> print(env.action_space.low)
[-1. -1. -1. -1.]

>>> print(env.action_space.high)
[1. 1. 1. 1.]
```

Això vol dir que cada acció ve definida per un array de quatre dimensions, on cada component pren un valor del domini dels nombres reals dins el rang [-1,1].

D'acord a la documentació disponible en el codi font de l'entorn[18], aquestes 4 components es corresponen als següents efectors, en aquest ordre:

[17] https://github.com/openai/gym/blob/master/gym/envs/box2d/bipedal_walker.py
[18] https://github.com/openai/gym/blob/master/gym/envs/box2d/bipedal_walker.py

- Força angular o torsió (*torque*) a aplicar sobre el maluc (*hip*) de la primera cama
- Força angular o torsió (*torque*) a aplicar sobre el genoll (*knee*) de la primera cama
- Força angular o torsió (*torque*) a aplicar sobre el maluc (*hip*) de la segona cama
- Força angular o torsió (*torque*) a aplicar sobre el genoll (*knee*) de la segona cama

Aquestes forces representen les que s'efectuarien en els motors de les articulacions (*joints*) del robot.

4.1.4 La Recompensa (Reward)

La recompensa consisteix en un valor proporcional al desplaçament efectual pel *walker*, més elevat com més ràpid es desplaci.

A aquest valor, però, se li resta una part proporcional a la torsió efectuada en els motors de les articulacions (joints), de manera que les polítiques que apliquen menor força als motors reben menor penalització.

També es penalitza el fet de no mantenir el cap recte.

Finalment, si el robot cau a terra, rep una recompensa de -100.

4.1.5 Els episodis

L'entorn considera que un episodi ha finalitzat en els següents casos:

- Quan s'aconsegueixen 300 punts en 1600 time steps en l'entorn *BipedalWalker-v3*, o 300 punts en 2000 time steps en l'entorn *BipedalWalkerHardcore-v3*.
- Quan el robot cau a terra.

Quan l'entorn considera que un episodi ha arribat a un estat terminal la funció **step** retorna el valor *False* per a la component *done* de la tupla de retorn.

4.2 Consideracions

Com es pot observar, començar a implementar algorismes d'RL en aquesta plataforma és molt senzill. Això la fa ideal per poder focalitzar-se en la part algorísmica del problema i poder comparar diferents solucions entre diferents equips de recerca.

5 Proposta de solució a desenvolupar

5.1 Enfocament emprat i justificació d'aquest

5.1.1 Proposta metodològica

El problema plantejat és un *Partially Observable Markov Decision Problem* (*POMDP*) amb espais d'estats i d'accions continus. Si bé per simplicitat es podria discretitzar l'espai d'accions, era tot un repte aventurar-se amb els algorismes d'RL per als espais continus d'accions, així que es va optar per aquesta opció.

En el moment de plantejar la solució, els principals algorismes candidats detectats (no evolutius) eren el PPO (*policy-based*), el DDPG (actor-crític basat en els DPG), el NAF (actor-crític basat en les DQN i adaptat per a espais continus d'accions) i la variant de NAF amb model. De manera que durant l'anàlisi, en funció de les possibilitats que ofereixen les biblioteques existents per a RL, es va determinar l'algorisme de base a implementar, que va ser el NAF.

Si bé en l'estat de l'art s'ha mencionat també els algorismes SAC (*Soft Actor Critic*) i CAQL (Continuous Action Q-Learning), malauradament aquests es van detectar posteriorment a l'elaboració d'aquest informe sobre l'estat de l'art, quan ja s'estava implementant la solució, de manera que no van ser incorporats en el disseny. Malgrat tot, sí que es va poder provar SAC amb la implementació pròpia de la biblioteca emprada per al desenvolupament del producte.

La proposta metodològica incloïa que, a poder ser, s'implementés també un agent amb algun altre algorisme, però això finalment no va ser possible per manca de temps.

Quant a l'input de l'agent, seguint la metodologia emprada per (Mnih, et al., 2013) per a les DQN aplicades als videojocs ATARI, es va proposar prendre com a observació la concatenació de 4 observacions consecutives de l'entorn. Això permet a l'agent tenir una noció de l'evolució temporal de l'entorn. Així mateixv, l'acció escollida per l'agent, s'executarà durant els següents 4 *time steps*.

Quant a la política d'exploració, es proposava comparar l'ús d'una estratègia €-*greedy* vers l'ús d'una funció de predicció del risc de caiguda, com es fa en l'algorisme del *Safe Reinforcement Learning (SRL)*. Per manca de temps, no va ser possible. En lloc seu, sí que es va poder provar una política d'exploració basada en l'addició de soroll, gaussià o brownià, sobre la política d'explotació.

Així mateix, per simplicitat, inicialment es va proposar obtenir una política determinista, però a mesura que es va anar adquirint destresa amb la biblioteca, es va poder acabar obtenint una política estocàstica, que per als entorn POMDP és preferible, ja que evita duar a terme sempre la mateixa acció en aquells estats percebuts idèntics però que corresponen a estats diferents de l'entorn.

Finalment, també en el cas de disposar de temps es proposava provar una metodologia basada en models per a la planificació per tal d'accelerar l'aprenentatge; malauradament, tampoc ha estat possible per les limitacions temporals del TFM.

5.1.2 Proposta tecnològica

Existeixen múltiples biblioteques que implementen diferents de les metodologies explicades en l'estat de l'art d'aquest TFM (Taula 8. Llistat de biblioteques per a Reinforcement Learning. Adaptat de Taula 8). La majoria d'elles amb *ports* per a Python i utilitzables en combinació amb la plataforma *OpenAI Gym*.

Biblioteca	URL del projecte
coach	https://github.com/IntelLabs/coach
dopamine	https://github.com/google/dopamine
Keras-RL	https://github.com/keras-rl
OpenAI Baselines	https://github.com/openai/baselines
OpenAI Spinning Up	https://github.com/openai/spinningup
RLlib	https://github.com/ray-project/ray/tree/master/rllib
Stable Baselines	https://github.com/hill-a/stable-baselines
TensorForce	https://github.com/tensorforce/tensorforce
TF-Agents	https://github.com/tensorflow/agents
TRFL	https://github.com/deepmind/trfl

Taula 8. Llistat de biblioteques per a Reinforcement Learning. Adaptat de (Shin, Cho, Jeon, Yoon, & Kim, 2019)

En l'article *On Choosing a Deep Reinforcement Learning Library* (Simonini, 2020) del blog tecnològic de l'empresa *data iku* es recull una anàlisi detallada d'algunes d'aquestes biblioteques que pot ser d'interès en la fase d'anàlisi i disseny de la solució. Es proposa escollir una de les biblioteques analitzades en el blog.

Així mateix, es proposa també utilitzar l'eina TensorBoard[19] per monitoritzar el procés d'aprenentatge i obtenir així gràfics que facilitin l'anàlisi dels resultats. Caldrà doncs aprendre com s'utilitza aquesta eina i com integrar-la en el problema en qüestió.

5.2 Requisits de la solució

5.2.1 Funcionals

- L'aplicació ha de mostrar una progressió positiva en l'aprenentatge del control autònom de la locomoció bípeda dels entorns BipedalWalker d'OpenAI Gym.
- L'evolució de l'aprenentatge s'ha de poder monitoritzar online, a mesura que aquest es va produint, preferiblement amb l'eina TensorBoard.

5.2.2 No Funcionals

- L'aprenentatge s'haurà de dur a terme en un temps d'execució raonable (idealment d'unes poques hores, millor que no pas dies) amb una màquina domèstica de prestacions convencionals.
- La màquina en el qual s'executarà l'aplicació disposa de tecnologia CUDA, amb la qual cosa se'n pot fer ús per tal d'accelerar els còmputs.
- L'aplicació, idealment, s'haurà de poder pausar i reprendre, atès que els períodes d'entrenament poden arribar a ser molt llargs.
- Idealment, s'ha de poder visualitzar gràficament l'entorn quan es desitgi i s'ha de poder anular aquesta opció per millorar el rendiment computacional.

[19] https://www.tensorflow.org/tensorboard

6 Anàlisi i disseny de la solució

6.1 Elecció de la biblioteca de desenvolupament: TF-Agents

L'elecció de l'algorisme ha estat un punt crític en el desenvolupament del projecte i ha tingut repercussions en el temps de desenvolupament.

Per a la seva elecció s'ha analitzat en primer lloc la disponibilitat dels diferents algorismes en les diferents biblioteques de Reinforcement Learning més habituals per a Python, així com el grau de documentació i de flexibilitat d'aquestes biblioteques, tot emprant com a punt de partida de l'anàlisi l'article *On Choosing a Deep Reinforcement Learning Library* del blog tecnològic de l'empresa *data iku* (Simonini, 2020).

En aquest sentit, les principals biblioteques candidates han estat Stable Baselines, per la seva documentació de gran qualitat, i Tensorflow Agents, per la seva elevada modularitat i flexibilitat en la composició dels algorismes.

Atès que un dels objectius d'aquest treball final de màster és el de poder combinar diferents tecnologies, es va decidir finalment optar per la biblioteca Tensorflow RL Agents, donada la gran modularitat.

6.1.1 TensorFlow Agents (TF-Agents)

TF Agents és una biblioteca de components per a *Reinforcement Learning* desenvolupada per sobre de la biblioteca Tensorflow. És un producte en el desenvolupament del qual hi participen alguns enginyers de Google Brain i, malgrat no ser un producte oficial de TensorFlow, si que s'ofereix directament des del seu lloc web.

La biblioteca disposa de la seva API i conté una petita documentació i un conjunt de tutorials per iniciar-s'hi.

Així mateix, al llarg del desenvolupament del projecte, també ha estat de molta utilitzat el lloc web de *Path to Pioneer* (Holt, 2020), que conté múltiples tutorials i codis d'exemple per a la implementació dels agents.

6.1.1.1 Arquitectura de TF-Agents

L'arquitectura bàsica de TF-Agents es basa en el model de components que es mostra en la Figura 16, adaptada de la documentació proporcionada en el lloc web oficial[20] d'aquesta.

En concret, els agents encapsulen una o més polítiques (d'explotació i d'exploració) i, si l'algorisme les utilitza, una o més funcions valor (ja siguin V-value, Q-value, A-value o variants), de les quals també en poden haver més d'una en funció de l'algorisme (la que s'aprèn, les *target* per al càlcul dels objectius, dobles, triples, adversàries, etc.). Tant les polítiques com les funcions valor poden estar formades per xarxes neuronals profundes.

[20] https://www.tensorflow.org/agents

L'agent interacciona amb l'entorn mitjançant un *Driver*, que utilitza una política de recol·lecció proporcionada per l'agent. El *driver* recopila un nombre configurable (*batch*) d'experiències (*time steps*) i les emmagatzema en una memòria intermèdia anomenada *Buffer Replay*.

Quantes experiències recopila el *driver* en cada iteració, quantes en presenta el *buffer* a l'agent, en quin ordre i quan de temps s'han de mantenir en el *buffer* són paràmetres que depenen de cada algorisme.

Els agents reben els *batchs* d'experiències com a paràmetre en la invocació del seu mètode *train*. Aquest mètode entrena les seves polítiques o funcions valors a partir d'aquestes experiències, i retorna un valor de pèrdua (*loss*) que és una mesura de l'error entre els valors estimats actualment per l'agent i els valors rebuts en el *batch* d'experiències.

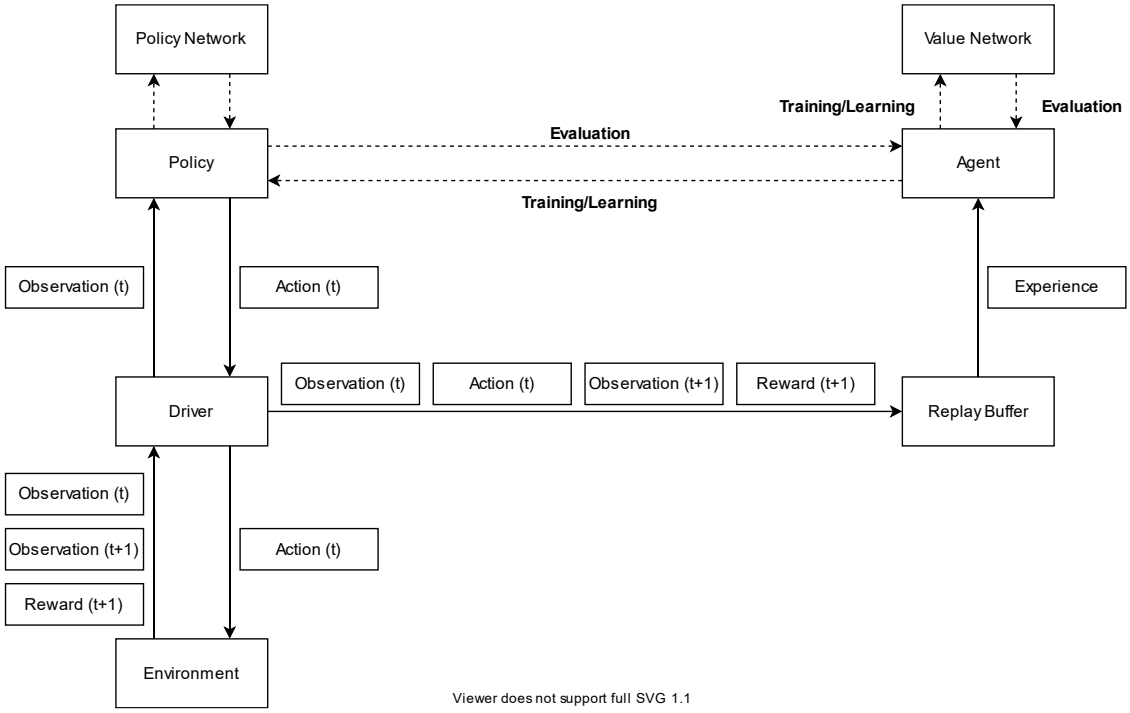


Figura 16. Arquitectura bàsica de l'aprenentatge per reforç amb TF Agents

La biblioteca conté agents ja codificats per als algorismes DQN, DDQN, DDPG, PPO, Reinforce, SAC i TD3.

6.1.2 TensorFlow

Tensorflow[21] (Google Research, 2015) és una biblioteca de codi obert desenvolupada per *Google Research* i especialitzada en *Machine Learning*. El seu nom prové del mot *Tensor*, que és el nom que rep la seva estructura de dades principal.

Figura 17. Logotip oficial de TensorFlow

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

Un *Tensor* és un array multidimensional tipat (Figura 18), amb una sèrie de propietats (com la seva forma o *shape*) i que és tractat pels elements de la biblioteca de manera molt eficient.

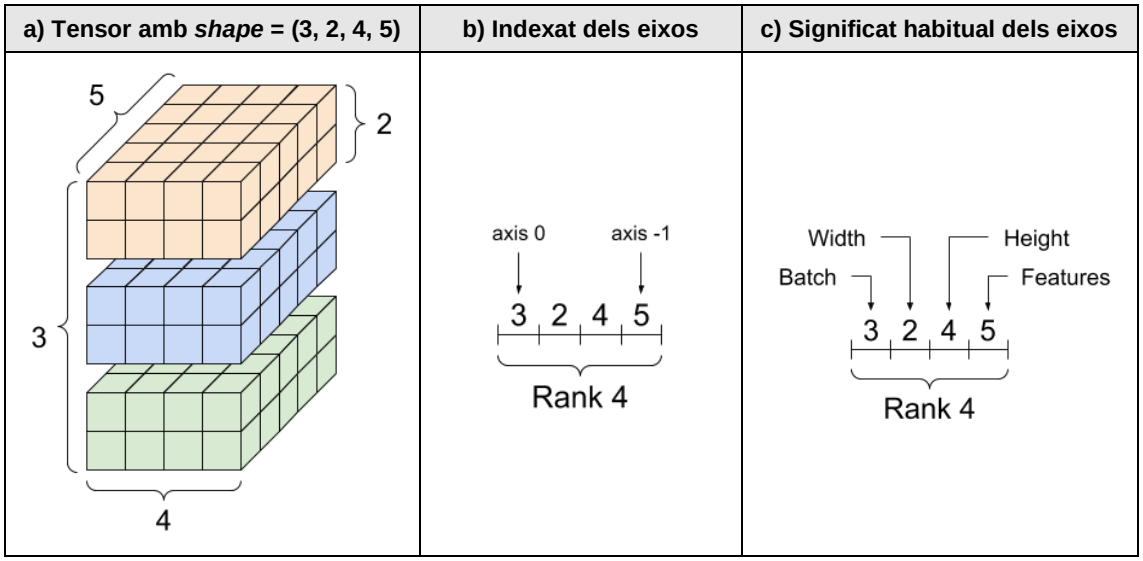


Figura 18. Exemple de Tensor multidimensional. Extret de https://www.tensorflow.org/guide/tensor

D'altra banda, TensorFlow permet organitzar les operacions a executar com a nodes d'un graf dirigit a les fulles del qual hi ha els tensors origen el còmput i, a l'altre extrem del graf, els tensors resultants del còmput (Figura 19). A aquest pas dels tensors d'un node als següents se'l coneix com a flux (*flow*) i és el mot que combinat amb el de tensor origina el nom complet de la biblioteca (*TensorFlow*).

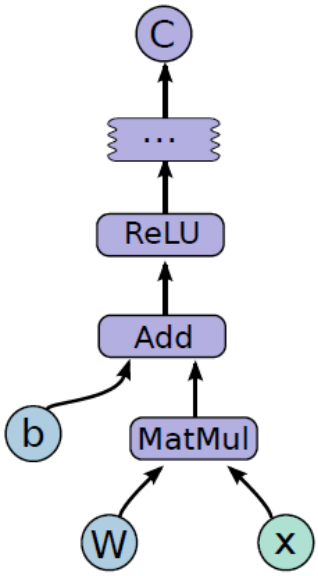


Figura 19. Exemple de Graf a TensorFlow. Extret de (Google Research, 2015)

Aquesta estructura permet construir el flux del tractament i executar-lo a continuació, quan sigui necessari. En aquest sentit, alguns elements, com la *shape* dels elements, podran tenir una definició estàtica (en el moment de crear el graf) i una de dinàmica (en el moment d'executar-se). Per això alguns components es poden declarar amb dimensions desconegudes (si bé la declaració és necessària).

Tanmateix, aquesta manera de desenvolupar resulta en dificultats per a la depuració del codi. En aquest sentit, la biblioteca permet activar el mode d'execució "impacient" (*eager execution*)[22], el qual avalua les expressions de forma immediata, sense construir grafs, i permeten conèixer en tot moment l'estat, estructura i continguts dels tensors. Durant el desenvolupament del projecte s'ha hagut de fer ús d'aquest mode, però s'ha vetllat també per tal que el codi s'executés també en el mode de graf, com es veurà més endavant (en concret, els mètodes *train* dels agents i els mètodes *collect* dels *divers* s'executen en mode graf).

Entre els principals avantatges de TensorFlow destaquen:

- **Portabilitat del codi**: implementacions (*kernels*) de les seves funcions per a diferents sistemes (GPU, CPU i altres dispositius).
- **Elevat rendiment**: la biblioteca és capaç d'optimitzar enormement les funcions a executar (grafs, compartició de *kernels*, paral·lelisme, computació distribuïda, etc.).
- **Diferenciació automàtica**: la biblioteca permet observar determinades variables en relació conjunts operacions concretes les quals s'enregistren durant la seva execució (*forward pass*). Finalitzada l'execució del bloc corresponent, es calculen els gradients per a cadascuna de les variables en relació al procediment enregistrat tot recorrent el graf en sentit invers al d'execució (*backward pass*) (Figura 20).

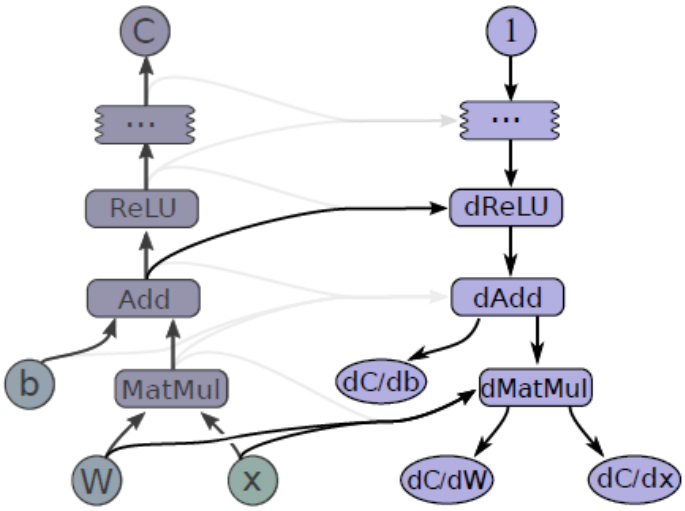


Figura 20. Exemple de diferenciació automàtica a TensorFlow. Extret de (Google Research, 2015)

- Disponibilitat de moltes altres **biblioteques de més alt nivell** d'abstracció per sobre de TensorFlow, com *keras*, que implementa una gran quantitat de funcionalitats relacionades amb les xarxes neuronals.
- **Elements per al monitoratge** del progrés i de les mètriques, com el mòdul *summary*, que permet crear fitxers de *log* amb les mesures de les mètriques i/o dades d'interès que podran ser posteriorment analitzades amb l'eina TensorBoard.

[22] Per a més informació veure: https://www.tensorflow.org/guide/eager

6.1.3 Tensorboard: una eina per al monitoratge dels resultats

Tensorboard és un kit d'eines web per a la monitorització i visualització dels procediments de *Machine Learning* de TensorFlow.

Combinat amb les mètriques ja implementades per la biblioteca *keras*, i fent ús del mòdul *summary* de TensorFlow, es pot anar creant un fitxer de *log* amb les dades de monitoratge i visualitzar-les a temps real amb l'eina.

En concret, TensorBoard es distribueix com un servei executable a la màquina local que crea un servidor web que escolta per defecte al port 6006 de l'adreça IP de localhost.

L'aspecte que té l'eina es mostra a la Figura 21.

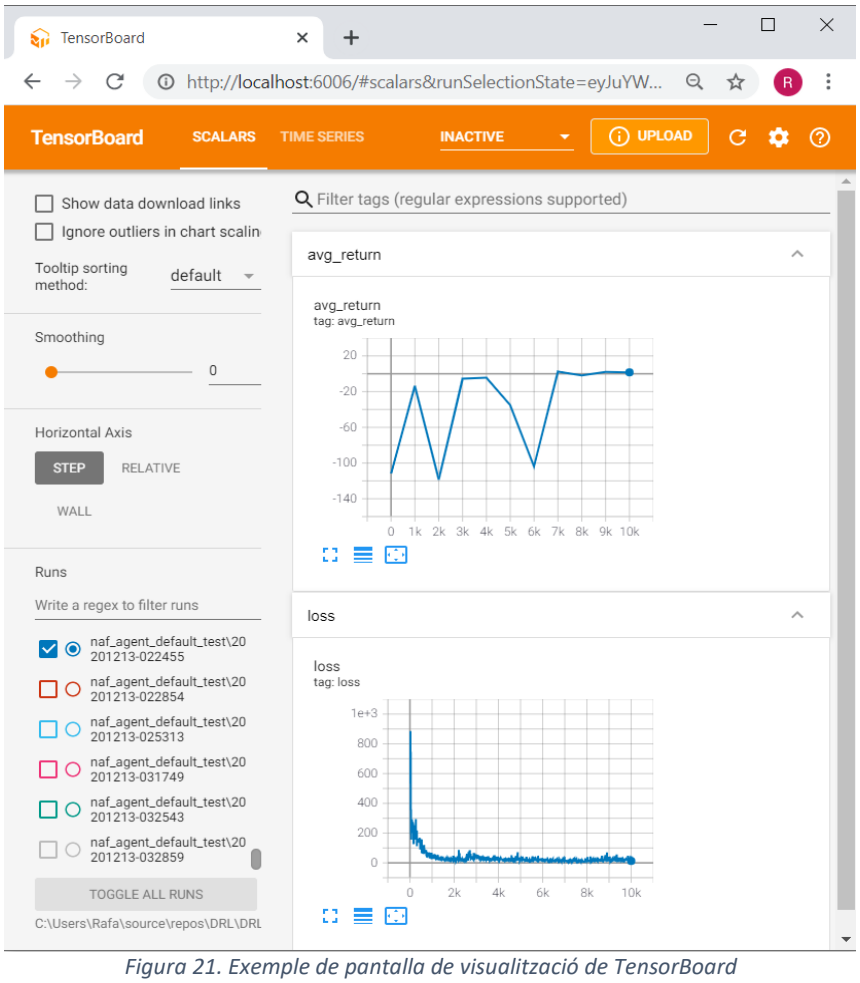


Figura 21. Exemple de pantalla de visualització de TensorBoard

6.2 Algorisme a implementar: Normalized Advantage Function (NAF)

Havent analitzat l'estat, s'ha decidit implementar l'algorisme NAF (Gu, Lillicrap, Sutskever, & Levine, 2016) ateses les següents consideracions:

- L'algorisme accepta espais d'observacions i d'accions continus, de manera que no seria necessari discretitzar l'espai d'accions.
- L'algorisme encara no es troba implementat en la biblioteca TF Agents, la qual cosa aporta valor afegit al Treball Final de Màster.
- L'estructura de l'algorisme és prou senzilla com per poder ésser implementada dins els límits temporals del projecte, malgrat ha estat necessari profunditzar en alguns aspectes ja que no es tenia prou coneixement sobre com implementar-los a l'inici de la fase d'implementació.
- L'algorisme sembla força eficient per poder ser executat en un ordinador domèstic, ja que entrena la política i les funcions valor a la vegada en la mateixa passada de cada iteració.
- La simplicitat de l'algorisme fa més senzill aplicar-hi modificacions per integrar-lo amb altres metodologies, com la del *Safe Reinforcement Learning*.

L'Algorisme 8 mostra el pseudocodi de l'algorisme NAF, on x representa una observació de l'entorn i u l'acció a executar.

1. Inicialitzar a l'atzar la xarxa Q normalitzada $Q(x,u|\theta^Q)$
2. Inicialitzar la xarxa d'objectius (target) Q' amb els pesos de Q: $\theta^{Q'} \leftarrow \theta^Q$
3. Inicialitzar el replay buffer $R \leftarrow \emptyset$
4. **Per cada** episodi **fer:**
5. Inicialitzar un procés aleatori $\mathcal{N}$ per a l'exploració d'accions
6. Obtenir l'estat inicial x_1
7. **Per cada** step t de l'espisoi **fer:**
8. **#Exploració:**
9. Seleccionar una acció $u_t = \mu(x_t|\theta^\mu) + \mathcal{N}_t$
10. Executar l'acció u_t i observar r_{t+1} i x_{t+1}
11. Desar la transició $(x_t, u_t, r_{t+1}, x_{t+1})$ en la memoria R
12. **#Experience replay:**
13. **Repetir** J **vegades:**
14. Seleccionar un conjunt I de m transicions a l'atzar de la memoria R
15. **Per cada** transició T_i de I **fer:**
16. **#Calcular els targets:**
17. **Si** T_i és terminal **llavors:**
18. $y_i = r_i$
19. **Sinó:**
20. $y_i = r_i + \gamma V'(x_i'|\theta^{Q'})$
21. **FiSi**
22. **FiPer**
23. **#Estimació de Q:**
24. Calcular la pèrdua (loss) $L = \frac{1}{N}\Sigma_i\left(y_i - Q(x_i, u_i|\theta^Q)\right)^2$
25. Fer un descens de gradient per actualitzar θ^Q respecte a L
26. Actualitzar la xarxa d'objectius $\theta^{Q'} = \tau\theta^Q + (1-\tau)\theta^{Q'}$
27. **FiRepetir**
28. **FiPer**
29. **FiPer**

Algorisme 8. Continuous Q-Learning mitjançant NAF, adaptat de (Gu, Lillicrap, Sutskever, & Levine, 2016)

En aquest algorisme, com en les *dueling DQN*, la funció valor-acció (Q) s'obté de l'addició de la funció avantatge-estat (A) a la funció valor-estat (V) (Expressió 46).

$$Q(x,u|\theta^Q) = A(x,u|\theta^A) + V(x|\theta^V) \qquad \text{Expressió 46}$$

A la vegada, la funció avantatge es defineix d'acord a l'Expressió 47.

$$A(x,u|\theta^A) = -\frac{1}{2}\big(u-\mu(x|\theta^\mu)\big)^T P(x|\theta^P)\big(u-\mu(x|\theta^\mu)\big) \qquad \text{Expressió 47}$$

Aquesta formulació permet garantir un resultat sempre negatiu o zero si P és una matriu quadrada definida positiva, ja que aleshores es compleix l'Expressió 48[23].

$$P\ és\ definida\ positiva\ \Leftrightarrow \boldsymbol{w}^T P\boldsymbol{w} \in \mathbb{R}^+, \forall \boldsymbol{w} \in (\mathbb{R}^{\boldsymbol{n}} - \{\boldsymbol{0}\}) \qquad \text{Expressió 48}$$

Aquesta matriu P, a la vegada, es defineix com a depenent de l'estat x i caracteritzada per un conjunt de paràmetres θ^P.

Matemàticament es pot obtenir una matriu definida positiva aplicant l'Expressió 49.

$$P(x|\theta^P) = L(x|\theta^P)L(x|\theta^P)^T \qquad \text{Expressió 49}$$

on L és una matriu triangular inferior els elements de la qual són proporcionats per una xarxa neuronal amb paràmetres θ^P, i on els elements per a la diagonal fan d'exponent del nombre e (forçosament positius i més grans que zero).

D'aquesta manera, en l'Expressió 47 el que s'està duent a terme és, en primer lloc, calcular la el vector diferència entre l'acció executada en una experiència prèvia i la que preveu la política μ actual; en segon lloc, fent ús de la matriu $P(x|\theta^P)$, aplicar l'Expressió 48 per obtenir un resultat forçosament positiu o zero; finalment, en multiplicar el resultat per l'escalar $-1/2$, s'obtindrà el resultat de la funció avantatge, que serà forçosament negatiu o zero. En concret, l'avantatge només podrà ser zero quan l'acció que predigui la política sigui igual a l'acció executada anteriorment, fet que té tot el sentit, ja que en estar buscant la política òptima, això indica que l'acció era l'acció òptima.

Per al cas de l'entorn BipedalWalker2D d'OpenAI Gym, l'espai d'accions és de tipus $\mathbb{R}^4$, ja que només té 4 efectors. Això vol dir que la matriu P haurà de tenir unes dimensions de 4x4 i, per tant, el mateix passarà amb la matriu L.

El nombre d'elements d'una matriu triangular es pot calcular com el sumatori de la progressió aritmètica de l'1 fins al nombre d'elements de la diagonal, és a dir, fins al nombre d'elements de l'espai d'accions (Expressió 50).

$$S_n = n(1+n)/2 \qquad \text{Expressió 50}$$

Per tant, per al cas del BipedalWalker2D, el nombre d'elements necessaris per a la matriu L és de 10 i, per tant, aquesta serà el nombre d'unitats de la sortida de la xarxa neuronal caracteritzada per θ^P.

[23] Una matriu P és definida positiva si i només si el producte $\boldsymbol{w}^T P\boldsymbol{w}$ dona com a resultat un escalar positiu (més gran que zero) per a qualsevol vector $\boldsymbol{w}$ diferent del vector **0**.

6.3 Decisions de disseny

En TF-Agents cal distingir entre la implementació de l'agent i l'algorisme d'RL.

6.3.1 Disseny de l'agent

Els agents de TF-Agents hereten de la classe *TFAgent* i, mitjançant el mètode *train*, reben un conjunt (*batch*) d'experiències provinents del *buffer replay* i les utilitzen per aprendre i generar la política òptima. La classe que representa l'agent NAF a desenvolupar s'ha anomenat NafAgent.

Seguint la forma com s'estructura la biblioteca TF-Agents, aquesta classe s'ha ubicat dins la jerarquia de mòduls *agents* > *naf* > *naf_agent* (Figura 22). Així mateix, el mòdul *naf* contindrà també totes aquelles classes i funcions d'utilitat per a la creació i execució d'aquest tipus d'agents, com poden ser les classes que defineixen les xarxes neuronals que els seran necessàries.

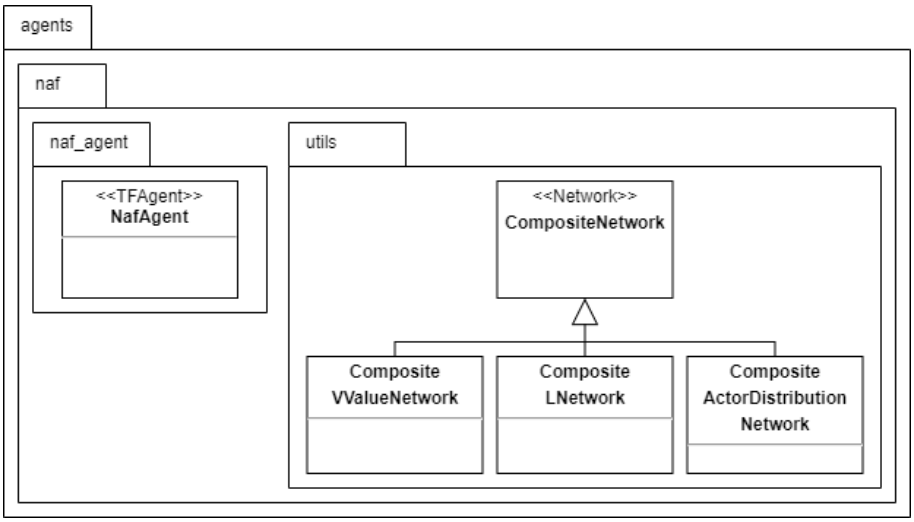


Figura 22. Components del mòdul desenvolupat en referència a l'agent NAF

En concret, l'agent NafAgent haurà d'implementar el mètode privat *_train* que és el que crida la superclasse TFAgent quan s'invoca el seu mètode públic *train*.

A més, durant la seva instanciació, l'agent haurà de generar les polítiques d'explotació i d'explotació que requereix el constructor de la superclasse TFAgent.

Per poder generar aquestes polítiques i configurar tot el procés d'aprenentatge, el constructor de l'agent NAF rebrà els següents paràmetres:

- *time_step_spec*: la definició dels espais d'observacions i de recompenses
- *action_spec*: la definició de l'espai d'accions
- *v_network*: la xarxa neuronal per a l'aproximació a la funció valor-estat
- *l_matrix_network*: la xarxa neuronal per generar els termes de la matriu L
- *policy_network*: la xarxa neuronal per definir la política d'actuació
- *optimizer*: la funció d'optimització que es desitja que utilitzi l'agent per a l'aprenentatge de les seves xarxes neuronals
- *target_update_tau*: el factor d'actualització de la xarxa *target*

- *target_update_period*: cada quantes iteracions d'entrenament de l'agent es vol que s'actualitzi la xarxa *target*
- *td_errors_loss_fn*: la funció que es vol emprar per al càlcul de la diferència temporal (l'algorisme NAF utilitza l'error quadràtic)
- *gamma*: el factor de decreixement de la influència de les recompenses futures
- *train_step_counter*: comptador del nombre d'iteracions d'entrenament que ha efectuat l'agent (es passa com a paràmetre per facilitar el monitoratge)
- *noise_factor*: grau de soroll que es vol afegir, es correspon a la desviació estàndard del soroll.
- *debug_summaries*: booleà per indicar si es vol un major grau de detall en les mètriques que es monitoritzen durant l'aprenentatge
- *summarize_grads_and_vars*: booleà per indicar si es vol monitoritzar també els gradients i variables obtinguts durant el procediment d'optimització.
- *name:* nom que se li vol donar a l'agent (d'ajuda al depurar)

En el capítol d'implentació es detallarà com s'ha implementat el constructor de la classe NafAgent així com el seu mètode *_train* i els diferents mètodes auxiliars necessaris per al seu funcionament.

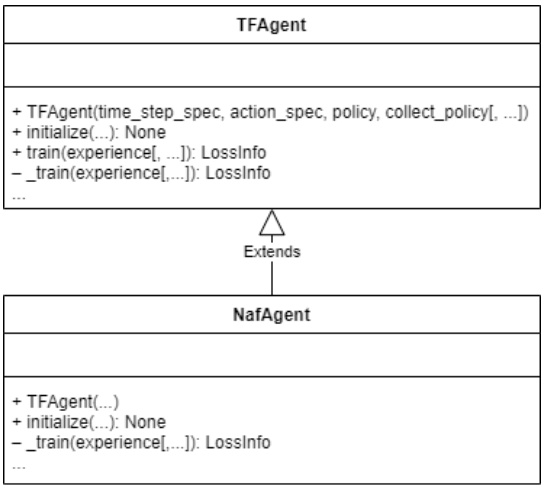


Figura 23. Relació entre la classe NafAgent desenvolupada i la classe TFAgent de la biblioteca TF-Agents

Disseny de les xarxes neuronals

En concret, l'estructura de les NAF consta de les següents xarxes neuronals:

- La "xarxa" $Q(x,u|\theta^Q)$: per al càlcul de la funció valor-acció Q. Aquesta, en realitat, es composa de les següents subxarxes:
    - $V(x|\theta^V)$: la xarxa per al càlcul de la funció valor-estat
    - $A(x,u|\theta^A)$: la xarxa per al càlcul de la funció avantatge-acció. Aquesta, a la vegada, es composa de les següents subxarxes:
        - $\mu(x|\theta^\mu)$: la política òptima en aprenentatge
        - $L(x|\theta^P)$: la xarxa per al càlcul dels termes de la matriu L, que servirà per calcular la matriu l'avantatge.

- La "xarxa *target*" $Q'(x,u|\theta^{Q'})$: per al càlcul dels objectius per a la funció valor-acció Q. Aquesta, en realitat, només s'empra per calcular el valor-estat per a les transicions en avaluació, ja que la conseqüència de l'acció s'obté a partir de la recompensa percebuda (cal recordar que la funció acció-valor és aquella en què es computa la conseqüència de l'acció executada i, a partir d'aquell moment, la conseqüència d'executar la política òptima, és a dir, el valor-estat). Com a conseqüència, no cal mantenir totes les subxarxes per a la xarxa *target*, sinó que només amb la xarxa $V'(x|\theta^{V'})$ seria suficient.

D'altra banda, aquestes xarxes es poden implementar totes elles de forma independent, o bé es pot partir d'una subxarxa (*preprocessing network*) comuna. També es poden combinar de diferents maneres, com per exemple, una subxarxa compartida entre les xarxes V i L, però que la xarxa de la política sigui independent. La Figura 24 esquematitza aquest darrer cas.

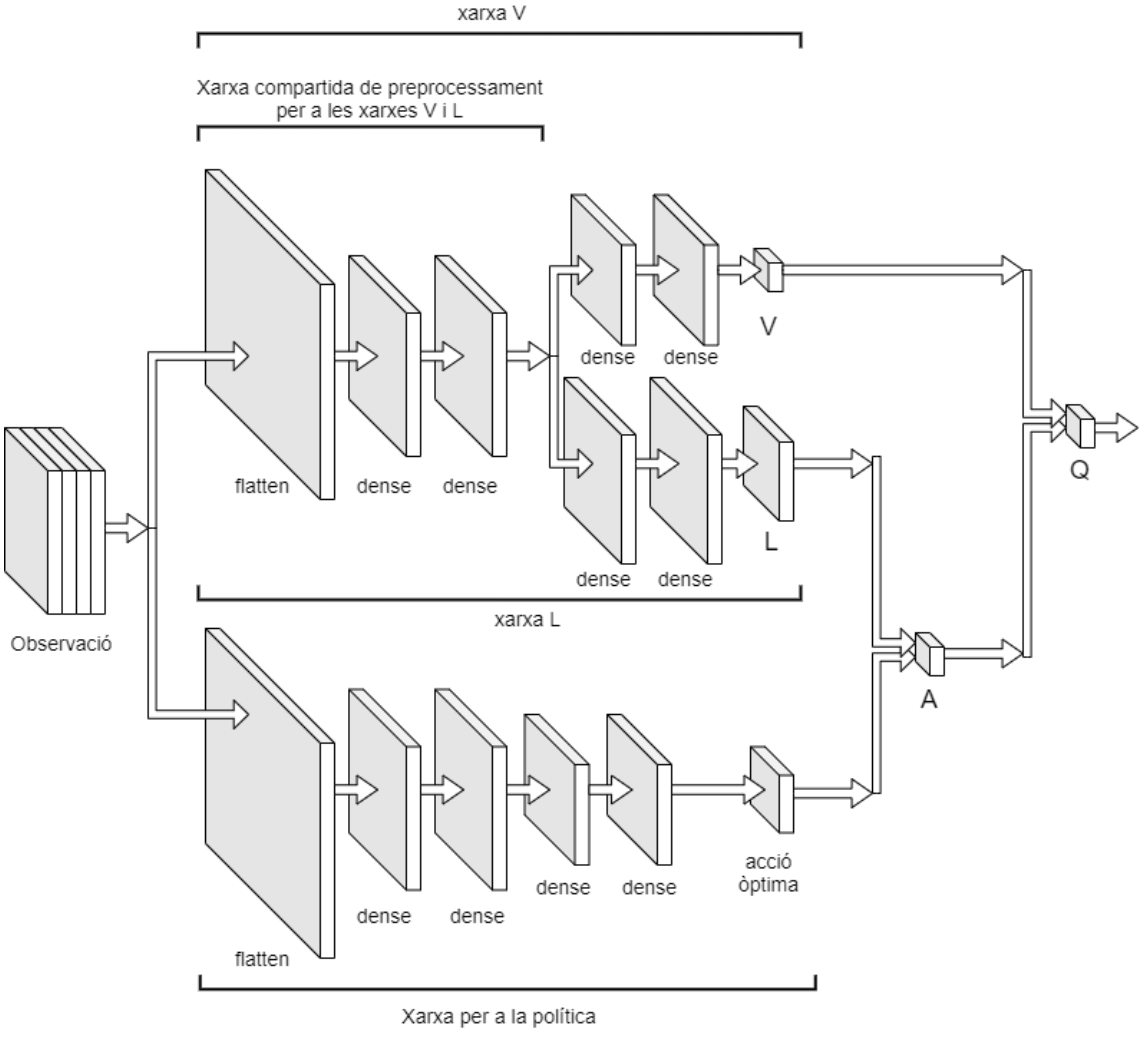


Figura 24. Exemple de cas de subxarxes compartides a NAF

Malauradament, a nivell de components, la biblioteca TF-Agents no incorpora l'opció de compartir una subxarxa. Sí que permet, en el moment de definir una xarxa *X*, proporcionar xarxa de preprocessament *P*, però en inicialitzar la xarxa *X* el que generen les xarxes predefinides de TF-Agents és una còpia independent *P'* de la xarxa *P*.

Per aquest motiu, entre les utilitats desenvolupades en el mòdul *naf* s'ha dissenyat també un tipus de xarxa que permet compartir una xarxa de preprocessament. A aquest tipus se l'ha anomenat *CompositeNetwork* i se n'ha derivat tres tipologies de xarxes emprades per NAF, que s'han anomenat CompositeVValueNetwork (per a la funció V), CompositeLNetwork (per a la funció L) i CompositeActorDistributionNetwork (per a la política).

La Figura 22 esquematitza aquesta jerarquia de components.

Disseny de les polítiques

El disseny de les polítiques a TF-Agents resulta molt senzill. La biblioteca incorpora una política per defecte anomenada ActorPolicy la qual rep com a paràmetre en la seva instanciació la xarxa neuronal que es vol que utilitzi per determinar l'acció a executar. El fet que aquesta política sigui estocàstica o determinista bé definit per la pròpia xarxa neuronal, en funció de si hereta del tipus ActorDistributionNetwork o no, respectivament.

L'agent, rep com a paràmetre del seu constructor la xarxa neuronal a utilitzar per a la política, així que durant la seva instanciació genera les polítiques.

La política d'explotació és una instància d'ActorPolicy, amb la xarxa neuronal proporcionada per a la política.

Per a les polítiques d'exploració, la biblioteca TF-Agents també ofereix diferents utilitats, entre les quals destaquen els PolicyWrappers. Aquests són embolcalls que es poden posar a la política d'explotació de manera que, sense modificar la política original, afegeixen algun tipus d'exploració aleatòria.

Entre aquests wrappers destaquen els següents:

- **EpsilonGreedyPolicy**: aplica una política d'exploració ϵ-*greedy*, de manera que amb una probabilitat ϵ s'executarà una acció a l'atzar d'entre les possibles dins l'espai d'accions i, amb una probabilitat 1-ϵ s'executarà l'acció determinada per la política d'explotació.

- **GaussianPolicy**: permet afegir soroll gaussià al voltant de l'acció a executar. Aquest soroll ve definit pel paràmetre *scale*, que es correspon a la desviació estàndard de la gaussiana que defineix el soroll a afegir.

- **OUNoisePolicy**: permet afegir soroll brownià de tipus Ornstein Uhlenbeck (OU) al voltant de l'acció a executar. Hi ha diferents maneres d'implementar aquest soroll i la biblioteca TF-Agens ho fa mitjançant un mètode oscil·latori definit pels paràmetres *ou_stddev* i *ou_damping*. El paràmetre *ou_damping* controla les oscil·lacions (durant quantes iteracions es mantindrà un soroll derivat de l'afegit en un instant de temps). El paràmetre ou_stddev regula la intensitat del soroll, essent la desviació estàndard de la funció gaussiana que el generarà.

La proposta de disseny és que durant la implementació es provin els diferents mètodes i se seleccioni aquell que resulti més eficient.

6.3.2 Disseny de l'algorisme Continuous Q-Learning amb NAF en TF-Agents

Per a l'execució de l'algorisme se segueix l'arquitectura bàsica de TF-Agents vista en la Figura 16. Breument, aquesta arquitectura inclou l'**entorn** (proporcionat per la biblioteca OpenAI Gym), l'**agent** (el NAF agent o qualsevol altre que es vulgui provar), un **replay buffer** on emmagatzemar les experiències viscudes (si esdevenen més experiències que capacitat té el buffer, s'aniran eliminant les experiències més antigues), un **driver** que recol·lectarà noves experiències amb l'entorn a partir de la política d'exploració de l'agent, i un procediment d'obtenció de mostres d'experiències (**sampling**) i de crida al mètode d'aprenentatge (***train***) de l'agent.

Per al disseny d'aquest mòdul s'han analitzat els diferents exemples existents en la biblioteca TF-Agents per a cadascun dels agents que ja venen implementats[24].

En concret, en tractar-se NAF d'un algorisme amb *buffer replay*, *off-policy* i per a un espai d'accions continu, l'algorisme que més s'apropa a aquest paradigma és l'agent *SAC* (Soft Actor Critic). En aquest sentit, per al disseny del mòdul d'execució de l'algorisme només ha estat necessari adaptar el codi d'exemple de la biblioteca de l'agent *SacAgent* a l'algorisme NAF i el problema en qüestió.

Així mateix, s'ha aprofitat l'ocasió per executar l'experiment també mitjançant l'agent SAC que ja ve implementat en la biblioteca TF-Agents.

A aquest mòdul de parametrització i execució dels algorismes se l'ha anomenat *experiments*. Dins d'aquest s'ha creat un submòdul per cada algorisme a provar:

- *off_p_tsb_experiment*: conté la classe *OffPolicyTimeStepBasedExperiment*, la qual permet configurar i executar un experiment parametritzat basat en l'aprenentatge per *time steps*, *off-policy*, i amb *buffer replay*. Aquesta classe serveix com a base (superclasse) de la resta de classes d'experimentació. El seu mètode privat *_get_the_agent* és qui proporciona l'agent amb què experimentar i el mètode public *copy*, delegant al mètode privat *_copy*, permet generar objectes còpia de l'experiment; Les subclasses han de sobreescriure aquests dos mètodes privats. Per executar l'experiment s'utilitza el mètode *launch*. Tots els objectes que intervenen en un experiment s'instancien de nou a l'invocar *launch*, de manera que no es comparteix estat entre experiments. Els paràmetres de l'experiment són atributs públics de l'objecte.

- *rnd_experiment*: conté la classe *RndExperiment*, la qual genera i utilitza un agent amb una política d'accions aleatòries de l'espai d'accions. Aquest agent serveix com a línia de base per comparar l'èxit de la resta d'agents.

- *naf_experiment*: conté la classe *NafExperiment*, que permet configurar i executar experiments parametritzats amb un agent NAF.

- *sac_experiment*: que conté la classe *SacExperiment*, que permet configurar i executar experiments parametritzats amb un agent SAC.

[24] Cada agent ve amb un submòdul anomenat examples on hi ha un exemple d'execució de l'algorisme necessari per l'agent tant amb xarxes neuronals convencionals com amb xarxes neuronals recurrents.

Tots aquests mòduls, classes i mètodes s'esquematitzen en la Figura 25.

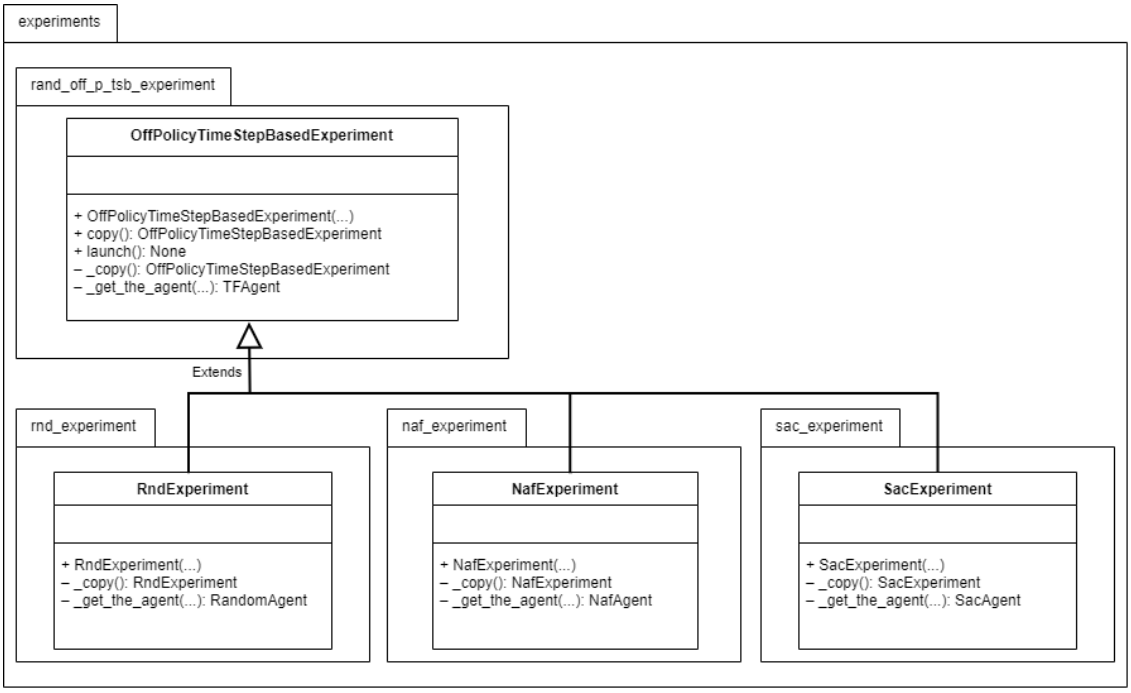


Figura 25. Estructura del mòdul d'experimentació

Paràmetres dels experiments

Durant la instanciació dels experiments cal passar-los com a paràmetres les seves característiques principals.

Paràmetres del constructor **OffPolicyTimeStepBasedExperiment**

- En relació a la **identificació de l'experiment**:
  - ***name:*** (cadena de text) nom per identificar l'experiment
- En relació al **sistema de fitxers**:
  - ***root_dir:*** (cadena de text) nom del directori base on emmagatzemar els registres de monitoratge, *checkpoints* i vídeos generats. Dins s'emmagatzemaran en un subdirectori de nom el *name* de l'experiment.
- En relació al **monitoratge**:
  - ***summary_interval:*** (enter) cada quantes iteracions de l'entrenament es volen desar les mètriques per tal de monitoritzar-les a TensorBoard.
  - ***log_interval:*** (enter) cada quantes iteracions de l'entrenament es vol mostrar per consola informació sobre l'evolució de l'algorisme.
  - ***debug_summaries:*** (booleà) si es desitja informació addicional sobre el càlcul de les mètriques.
  - ***summarize_grads_and_vars:*** (booleà) si es vol monitoritzar també el procediment d'optimització (variables i gradients corresponents)
- En relació a l'**entorn sobre el qual experimentar**:
  - ***env_name:*** (cadena de text) nom de l'entorn d'OpenAI Gym sobre el qual es vol experimentar, per exemple, "BipedalWalker-v3".
  - ***env_action_repeat_times:*** (enter) nombre d'observacions que es compactaran per passar a l'agent com a única observació, així com cops s'executarà en l'entorn l'acció seleccionada.

- En relació al procediment d'**avaluació de la política**:
  - ***eval_interval:*** (enter) cada quantes iteracions d'entrenament es vol avaluar la política en aprenentatge.
  - ***num_eval_episodes:*** (enter) quants episodis es volen executar per avaluar la política. La mètrica resultant en serà la mitjana aritmètica.
- En relació a la **recol·lecció d'experiències** durant l'experiment:
  - ***replay_buffer_capacity:*** (enter) quantes transicions es podran emmagatzemar en el buffer replay.
  - ***initial_collect_steps:*** (enter) quantes transicions es vol recollir mitjançant una política d'actuació aleatòria abans d'iniciar l'aprenentatge. Convé que sigui major que la ***sample_batch_size***.
  - ***num_iterations:*** (enter) quantes iteracions d'interacció amb l'entorn es volen executar.
  - ***collect_steps_per_iteration:*** (enter) quantes transicions es volen capturar en cada iteració d'interacció amb l'entorn.
- En relació al procés d'**aprenentatge**:
  - ***sample_batch_size:*** (enter) nombre de transicions, escollides a l'atzar del *replay buffer*, que es passen com a experiència prèvia a l'agent en cada iteració d'entrenament.
  - ***train_steps_per_iteration:*** (enter) nombre d'iteracions d'entrenament executa l'agent en cada iteració d'interacció amb l'entorn.
- En relació a l'**optimització dels càlculs** amb grafs de TensorFlow:
  - ***use_tf_functions:*** (booleà) si es vol utilitzar grafs de TensorFlow per obtenir millor rendiment o per contra, en el cas que sigui fals, utilitzar el mode "impacient" (*eager*), més lent però més fàcilment depurable.
- En relació a la creació de ***checkpoints***:
  - ***train_checkpoint_interval:*** (enter) cada quantes iteracions d'entrenament es crearà un *checkpoint* de l'agent i els seus components.
  - ***policy_checkpoint_interval:*** (enter) cada quantes iteracions d'entrenament es crearà un *checkpoint* de la política en aprenentatge.
  - ***replay_buffer_checkpoint_interval:*** (enter) cada quantes iteracions d'entrenament es crearà un *checkpoint* del *buffer replay*.

Paràmetres específics del constructor **RndExperiment**

Utilitza només els paràmetres definits per al constructor de la superclasse OffPolicyTimeStepBasedExperiment, a la qual en delega la inicialització.

Paràmetres específics del constructor **NafExperiment**

Utilitza els paràmetres definits per al constructor de la superclasse *OffPolicyTimeStepBasedExperiment* i, a més a més, els següents paràmetres:

- Per a la creació de les **xarxes neuronals**:
  - **Xarxa compartida** de preprocessament (opcional):
    - ***preprocessing_conv_layer_params:*** (tupla de tuples d'enters, opcional) estructura de les capes convolucionals d'acord a la definició d'aquestes capes en la plataforma *keras*.
    - ***preprocessing_conv_type:*** (cadena de text, opcional) "1d" per a convolucions unidimensionals i "2d" per a les bidimensionals.

        - ***preprocessing_fc_layer_params:*** (tupa d'enters, opcional) cada enter de la tupla representa el nombre d'elements d'una capa totalment connectada (*fully connected*).
        - ***preprocessing_dropout_layer_params:*** (tupla de nombres reals, opcional) només té sentit si s'ha definit el paràmetre anterior. Es correspon a la taxa de regularització de les capes totalment connectades per d'anul·lació aleatòria i puntual d'unitats de les capes durant l'entrenament.
    - **Xarxa V:**
        - ***v_network_fc_layer_params:*** (tupa d'enters) anàleg al paràmetre preprocessing_fc_layer_params però per a la xarxa neuronal per al valor-estat (V)
        - ***v_network_dropout_layer_params:*** (tupa d'enters, opcional) anàleg al paràmetre preprocessing_dropout_layer_params però per a la xarxa neuronal per al valor-estat (V)
    - **Xarxa L:**
        - ***l_network_fc_layer_params:*** (tupa d'enters) anàleg al paràmetre preprocessing_fc_layer_params però per a la xarxa neuronal per als termes de la matriu triangular L.
        - ***l_network_dropout_layer_params:*** (tupa d'enters, opcional) anàleg al paràmetre preprocessing_dropout_layer_params però per a la xarxa neuronal per als termes de la matriu triangular L.
    - **Xarxa per a la política:**
        - ***policy_network_fc_layer_params:*** (tupa d'enters) anàleg al paràmetre preprocessing_fc_layer_params però per a la xarxa neuronal per a la política.
        - ***policy_network_dropout_layer_params:*** (tupa d'enters, opcional) anàleg al paràmetre preprocessing_dropout_layer-_params però per a la xarxa neuronal per a la política.
        - ***policy_network_uses_shared_preprocessing_network:*** (booleà) si la política ha de fer ús també de la xarxa compartida de preprocessament.
- Per inicialitzar el **soroll** de la política d'exploració
    - ***noise_factor:*** (real) equival a la desviació estàndard de la funció gaussiana que genera el soroll a afegir a la política d'explotació per generar la política d'exploració.
- Per al **procés d'aprenentatge** de les xarxes
    - ***learning_rate*****:** (real) taxa d'aprenentatge a aplicar a la funció d'optimització de les xarxes neuronals durant l'aprenentatge.
- Per a l'**actualització de la *xarxa target***:
    - ***target_update_tau:*** (real) factor tau d'actualització de la xarxa *target*.
    - ***target_update_period:*** (enter) cada quantes iteracions d'aprenentatge de l'agent es desitja actualitzar la xarxa *target*.
- Per al càlcul de l'**error TD**:
    - ***gamma:*** (real) factor de descompte de la influència de les recompenses futures respecte al valor la funció Q en procés d'aproximació.
    - ***td_errors_loss_fn:*** (funció) funció a utilitzar per calcular l'error TD. En el cas de NAF és l'error quadràtic.

Paràmetres específics del constructor **SacExperiment**

A més a més dels paràmetres definits per al constructor de la superclasse OffPolicyTimeStepBasedExperiment, aquest constructor té definits també:

- Paràmetres equivalents als del constructor NafExperiment:
  - ***target_update_tau***, ***target_update_period***, ***td_errors_loss_fn*** i ***gamma***
- Paràmetres específics del constructor **SacExperiment**:
  - **Per a la generació de les xarxes neuronals**
    - Per a l'**actor**:
      - ***actor_fc_layers:*** (tupa d'enters) anàleg al paràmetre policy_network_fc_layer_params de NafExperiment. És la xarxa que utilitzarà la política.
    - Per al **crític**:
      - ***critic_obs_fc_layers***, ***critic_action_fc_layers*** i ***critic_joint_fc_layers:*** (cadascun és una tupla d'enters) anàlegs al paràmetre actor_fc_layers però per a la xarxa neuronal del crític.
  - Taxa d'aprenentatge de cada xarxa:
    - ***actor_learning_rate:*** (real) taxa d'aprenentatge per a l'optimitzador de la xarxa actora.
    - ***critic_learning_rate:*** (real) taxa d'aprenentatge per a l'optimitzador de la xarxa del crític.
    - ***alpha_learning_rate:*** (real) coeficient de regularització de l'entropia.
    - ***gradient_clipping:*** (real) cota per als gradients
  - Per al càlcul de l'**error TD**:
    - ***reward_scale_factor:*** (real) factor d'escala a multiplicar a les recompenses.

Instanciació de l'entorn d'OpenAI Gym

Per tal de beneficiar-se del rendiment de la biblioteca de TF-Agents és convenient convertir els entorns d'OpenAI Gym a instàncies de la classe *TFEnvironment*, que utilitza tensors de TensorFlow per als seus càlculs.

Per dur a terme aquesta conversió, TF-Agents disposa d'un wrapper anomenat *TFPyEnvironemnt*, que facilita aquesta conversió.

Així mateix, TF-Agents té integrada la biblioteca OpenAI Gym i permet instanciar-ne els entorns Python mitjançant la funció *load* del mòdul *environments.suite_gym*.

D'altra banda, seguint l'exemple proporcionat per la biblioteca per executar i avaluar SAC, s'utilitzarà en cada experiments dos entorns diferents, un per a l'entrenament de l'agent i un altre per a l'avaluació de la política obtinguda fins al moment. D'aquesta manera, l'avaluació es fa sobre un entorn net cada cop, inicialitzat a l'atzar, i no influeix sobre l'entorn on s'està produint l'entrenament.

A més a més, per disminuir l'efecte estocàstic en l'avaluació de la política, s'utilitzarà el wrapper *GreedyPolicy*, de manera que s'obtingui el valor més probable per a l'acció d'acord a la distribució òptima generada per la política estocàstica en entrenament.

Recol·lecció de mètriques

El mòdul *metrics.tf_metrics* de la biblioteca TF-Agents incorpora un seguit de classes ja implementades per tal de facilitar el càlcul de les mètriques d'interès optimitzades amb l'ús de tensors de TensorFlow. Algunes d'aquestes mètriques són *AverageReturnMetric*, *AverageEpisodeLengthMetric*, *EnvironmentSteps* i *NumberOfEpisodes*, que seran les que es registraran en aquest projecte.

Per utilitzar aquestes mètriques només cal crear-ne una instància i gestionar-la mitjançant les funcions corresponents, com les que s'expliquen tot seguit.

L'escriptura de les mètriques al sistema de fitxers es du a terme mitjançant un objecte de tipus *SummaryWriter*, una classe definida en el mòdul *summary* de la biblioteca TensorFlow. Per obtenir una instància de *SummaryWriter* el mòdul *summary* de TensorFlow facilita la funció *create_file_writer*.

Els objectes *SummaryWriter* permeten emmagatzemar traces de diferents tipus (escalars, imatges, àudio, histogrames i text). Per fer-ho, cal fixar el *writer* com a *SummaryWriter* per defecte de TensorBoard i aleshores utilitzar les funcions proporcionats pel propi mòdul *summary* per emmagatzemar-es. La funció de *summary* més emprada en aquest projecte és el mètode *summary.scalar*.

També la biblioteca TF-Agents incorpora mètodes per a facilitar el càlcul i emmagatzematge de les traces. Per exemple, el mètode *eager_compute* del mòdul *eval.metric_utils*, permet passar com a paràmetre un llistat de mètriques, un entorn *TFEnvironment*, una política, un objecte de *SummaryWriter* i alguns paràmetres més i genera i desa automàticament les traces demanades.

En el disseny de la traçabilitat de l'aprenentatge, seguint l'exemple disponible en la biblioteca TF-Agents per a l'execució i avaluació de l'agent SacAgent, s'ha utilitzat dos objectes de tipus SummaryWriter, un per a les mètriques de l'entrenament i l'altre per a les mètriques de l'avaluació, de manera que cadascun emmagatzema les traces en un subdirectori diferent del directori arrel on s'emmagatzemen les traces.

Visualització de les mètriques amb TensorBoard

Les traces emmagatzemades amb els objectes de tipus *SummaryWriter* es poden visualitzar mitjançant l'eina TensorBoard. Com que les traces es poden emmagatzemar a mesura que es van generant, podem anar monitoritzant el procés d'aprenentatge a temps real a mesura que es va executant.

L'aplicació TensorBoard es pot executar mitjançant la comanda

```
python3 -m tensorboard.main --logdir=...
```

on el paràmetre *logdir* indica el directori on hi ha les traces que es volen analitzar.

TensorBoard posarà en marxa un petit servidor web a través del qual es visualitzar les traces mitjançant un navegador web qualsevol a l'adreça:

http://localhost:6006

Desament i recuperació de l'estat en el cas d'aturar l'execució

Una altra utilitat interessant de TF-Agents és la possibilitat d'emmagatzemar de forma senzilla instantànies de l'estat de l'agent (incloent l'estat de les seves xarxes neuronals), de les polítiques o del buffer replay.

A aquestes instantànies se les anomena *checkpoints* i permeten recuperar l'estat en el cas que l'execució s'aturi.

La classe que permet crear objectes encarregats de la gestió d'aquests *checkpoints* es troba dins el mòdul *utils.common* de TF-Agents i s'anomena *Checkpointer*. Aquesta rep com a paràmetres principals una cadena de text indicant el directori on emmagatzemar el *checkpoint* i el conjunt d'objectes que s'hi volen emmagatzemar.

El mètode *initialize_or_restore* de l'objecte *checkpointer* permet inicialitzar el conjunt de fitxers en el directori per al manteniment dels *checkpoints* si no existeix encara aquesta estructura, o bé si ja hi havia algun *checkpoint* emmagatzemat, restaurar-lo per tal de continuar l'algorisme en el darrer estat emmagatzemat.

Vídeo d'exemple d'execució de la política

Els entorns d'OpenAI Gym disposen d'un mètode anomenat *render* que permet visualitzar gràficament l'evolució de l'entorn simulat i, a la vegada, obtenir el mapa de bits corresponent a cada *time step* en un vector *numpy*.

(Holt, 2020) mostra en un codi d'exemple com generar un vídeo a partir d'aquesta seqüència de vectors de forma molt senzilla mitjançant la biblioteca *imageio*.

S'adaptarà aquest petit codi, distribuït sota llicència creative commons (CC BY-NC-SA 4.0), per a l'ús en aquest projecte com a mètode privat de la superclasse *OffPolicyTimeStepBasedExperiment*.

6.3.3 Disseny de la funció principal (*main*)

Configuració i llançament dels experiments

La forma més senzilla de configurar els experiments és mitjançant la parametrització de la seva instanciació. Així doncs, es crearà una variable per a cada paràmetre, per si cal modificar-les, i s'instanciaran els objectes corresponent delegant el valor del paràmetres a aquestes variables.

Els diferents objectes que representen experiments concrets, es poden replicar amb el mètode *copy* i en les rèpliques es pot modificar els paràmetres desitjats tot assignant els valors desitjats als seus atributs corresponent.

Un cop definits els experiments, es poden executar directament un a un o, mitjançant la biblioteca *multiprocessing*, executar-los en paral·lel.

6.4 Adaptació de l'entorn BipedalWalker2D

Atès que els *Problemes de Decisió de Markov (MDP)* són els computacionalment menys costosos de resoldre, cal modificar l'entorn BipedalWalker2D per tal que s'aproximi el màxim possible a la *Propietat de Markov*. En aquest sentit, convé que cada observació que rebi l'agent sobre l'estat contingui informació suficient per poder decidir quina acció dur a terme, independentment del que hagi succeït abans del darrer estat percebut.

Així doncs, una captura estàtica de les variables de l'entorn en un moment donat no serà prou útil, ja que l'agent no disposarà d'informació suficient per fer-se una idea sobre la noció del temps (trajectòries, velocitats, acceleracions, etc.).

La solució que es presenta en aquest projecte està inspirada en l'article de (Mnih, et al., 2015) sobre l'aplicació de l'algorisme DQN sobre un simulador de la consola Atari. En aquests experiment, els investigadors proporcionaven a l'agent una seqüència de quatre *frames* (fotogrames) seguits i, en conseqüència, aplicaven la mateixa acció determinada per l'agent durant els 4 frames següents. Seguint la mateixa aproximació, l'entorn (*environment*) s'encapsularà dins d'un *wrapper* que aplicarà aquest tractament, rebent una acció a executar com a input i retornant un batch la informació i observacions dels *n time steps* següents. La recompensa retornada serà la suma de les *n* recompenses durant aquests *n time steps*, si bé altres criteris també podrien ser adients.

6.5 Disseny de les prestacions corresponents a requisits no funcionals

Entre els requisits no funcionals es van incloure les següents especificacions:

- L'aplicació, idealment, s'haurà de poder pausar i reprendre, atès que els períodes d'entrenament poden arribar a ser molt llargs.
- Idealment, s'ha de poder visualitzar gràficament l'entorn quan es desitgi, però també s'ha de poder anular aquesta opció quan es vol simplement centrar-se en l'aprenentatge.

Aquests tipus de requisits requereixen de l'ús d'interrupcions. La forma més senzilla és emprar listeners d'esdeveniments de teclat (per exemple, mitjançant la biblioteca *pynput*) i llurs corresponents *event handlers*.

El control de la visualització dels vídeos es pot dur a terme mitjançant un interruptor lògic en l'entorn. Un booleà que indiqui si cal renderitzar-lo o no, i que s'activi o inactivi quan es premi una tecla concreta.

El mateix podem fer amb la pausa i la represa de l'execució, que a més a més dels esdeveniments de teclat, requerirà també de les utilitats de la classe *Checkpointer* de TF-Agents.

7 Implementació en Python

7.1 Entorn de desenvolupament

Per al desenvolupament del producte s'ha utilitzat el següent entorn:

- **Entorn de desenvolupament:**
  - Microsoft Visual Studio Community 2017
- **Entorn d'execució:**
  - Python 3.7 (64 bits)
  - tensorflow 2.3.1
  - tensorflow probability 0.11.1
  - tf_agents 0.6.0
  - tensorboard 2.4.0
  - NVIDIA CUDA 10.1
  - cuDNN 7.6.5

Malgrat es va començar amb l'entorn Microsoft Visual Studio Community 2019, donats problemes de compatibilitat d'algunes biblioteques i versions de Python, es va preferir utilitzar l'entorn 2017, el qual va funcionar sense problemes.

L'entorn de desenvolupament, d'execució i de depuració s'ha provat amb el codi de mostra de (Holt, 2020) per a l'entorn OpenAI Gym *"Pendulum-v0"*. Malauradament no ha estat possible fer funcionar la biblioteca amb la interfície CUDA i s'ha hagut d'inhabilitar de la següent manera:

```
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "-1"
```

Havent realitzat aquesta modificació, l'entorn s'ha executat perfectament, però sense acceleració per GPU.

7.2 Adaptació de l'entorn OpenAI Gym per a la repetició d'accions

TF Agents utilitza els anomenats *Environment Wrappers* (embolcalls) per tal de modificar el comportament per defecte d'un entorn predefinit. Aquests embolcalls exposen la mateixa API que els entorns de base *PyEnviroment* i la biblioteca n'aporta uns quants de ja implementats. Entre aquests, de cara al problema d'aquest treball, destaquen els *wrappers ActionRepeat* i *HistoryWrapper*.

El *wrapper ActionRepeat* permet executar una mateixa acció múltiples vegades en l'entorn. Malauradament només retorna la darrera observació i no l'històric d'aquestes. El *wrapper HistoryWrapper*, en canvi, retorna l'històric de les darreres n observacions sol·licitades, però no permet executar n accions repetides sobre l'entorn.

Malgrat es podrien combinar els *wrappers* entre sí per tal d'obtenir el comportament desitjat, s'ha decidit implementar-ne un de propi basat en el codi d'aquests dos per tal d'aprendre com és el seu funcionament.

En concret, un *wrapper* en TF Agents hereta de la classe *PyEnvironmentBaseWrapper*, que delega tots els mètodes de l'API de la classe *PyEnvironment* a l'entorn que se li passa com a paràmetre al constructor de la classe. Les classes filles de *PyEnvironmentBaseWrapper* només cal que sobreescriguin els mètodes delegats per tal de modificar el comportament desitjat.

Al *wrapper* creat per modificar el comportament de l'entorn d'acord a les especificacions proposades en la solució se li ha assignat el nom de *ActionRepeatHistoryWrapper*.

En concret, s'ha aprofitat i adaptat el codi de les classes *ActionRepeat* i *HistoryWrapper* de la biblioteca TF-Agents. A continuació es mostra el codi més rellevant de la classe desenvolupada.

```
class ActionRepeatEnvWrapper(PyEnvironmentBaseWrapper):
  def __init__(self, env: PyEnvironment, times: types.Int):
    ...
    self._times = times                                 # Extret d'ActionRepeat
    self._history_length = times                        # Adaptat d'HistoryWrapper
    ...

  ...

  def _step(self, action):
    ...
    total_reward = 0                                    # Extret d'ActionRepeat
    self._observation_history.clear()                   # Extret d'HistoryWrapper

    for step_number in range(self._times):              # Extret d'ActionRepeat
      time_step = self._env.step(action)                # Extret d'ActionRepeat
      if time_step.is_first():                          # Adaptat d'HistoryWrapper
        self._observation_history.extend(
                  [self._zero_observation]*(self._history_length - 1))
        break
      self._add_history(time_step)                      # Extret d'HistoryWrapper
      total_reward += time_step.reward                  # Extret d'ActionRepeat
      if time_step.is_last():                           # Adaptat d'HistoryWrapper
        self._observation_history.extend(
                  [self._zero_observation]*(self._history_length -1 -step_number))
        break

    total_reward = np.asarray(total_reward,             # Extret d'ActionRepeat
                              dtype=np.asarray(time_step.reward).dtype)

    observation = tf.convert_to_tensor(                  # Extret d'HistoryWrapper
                                       self._observation_history)

    return ts.TimeStep(time_step.step_type,             # Extret d'ActionRepeat
                       total_reward,
                       time_step.discount,
                       observation)
```

7.3 Desenvolupament del l'Agent NAF

Per a la implementació de l'agent NAF s'ha creat la classe NafAgent, que com tot agent de la biblioteca TF-Agents hereta de la superclasse TFAgent.

```
from tf_agents.agents.tf_agent import TFAgent
...
class NafAgent(TFAgent):
  """A NAF Agent."""
  ...
```

El mètode constructor *__init__*

El constructor s'ha definit d'acord als paràmetres establerts en la fase de disseny.

```
...
import tensorflow as tf
from tf_agents.typing import types
from tf_agents.networks import network
from tf_agents.trajectories import time_step

class NafAgent(TFAgent):

  def __init__(
    self,
    time_step_spec: time_step.TimeStep,
    action_spec: types.NestedTensorSpec,
    v_network: network.Network,
    l_matrix_network: network.Network,
    policy_network: network.Network,
    optimizer: types.Optimizer,
    target_v_network: Optional[network.Network] = None,
    target_update_tau: types.Float = 1.0,
    target_update_period: types.Int = 1,
    td_errors_loss_fn: types.LossFn = tf.math.squared_difference,
    gamma: types.Float = 1.0,
    train_step_counter: Optional[tf.Variable] = None,
    noise_factor: types.Float = 0.1,
    debug_summaries: types.Bool = False,
    summarize_grads_and_vars: types.Bool = False,
    name: Optional[Text] = None,
    ):
    ...
```

La classe *TFAgent*, a la vegada, hereta de la classe *Module* de TensorFlow. Un *Module* és un contenidor d'objectes principalment de les tipologies de Tensorflow *Variable* i/o *Module*, com són les xarxes neuronals[25]. Aquest contenidor tindrà un nom que en facilitarà el monitoratge. La primera operació que du a terme el constructor de l'agent NAF és inicialitzar-se com a *Module*.

```
class NafAgent(TFAgent):
  def __init__(... policy_network ...):
    ...
    tf.Module.__init__(self, name=name)
    ...
```

[25] Per a més informació veure: https://www.tensorflow.org/api_docs/python/tf/Module

A continuació, cada paràmetre rebut pel constructor és tractat i emmagatzemat per l'agent, si s'escau, com a atribut propi d'aquest. A tall d'exemple:

```
class NafAgent(TFAgent):
  def __init__(... policy_network ...):
    ...
    self._policy_network = policy_network
    ...
```

El següent pas és generar la política d'explotació. Tal i com es va establir en la fase de disseny, aquesta serà un objecte de la classe ActorPolicy de TF-Agents.

```
from tf_agents.policies import actor_policy
...
class NafAgent(TFAgent):
  def __init__(... policy_network ...):
    ...
    policy = actor_policy.ActorPolicy(time_step_spec = time_step_spec,
                                      action_spec = action_spec,
                                      actor_network = self._policy_network,
                                      clip=True)
    ...
    super(NafAgent, self).__init__(... policy=policy ...)
   ...
```

La política d'exploració s'obtindrà embolcallant la política d'explotació amb un *policy wrapper*, ja sigui de tipus *EpsilonGreedyPolicy*, *GaussianPolicy* o bé *OUNoisePolicy*. A tall d'exemple, es mostra el cas del *wrapper GaussianPolicy*.

```
from tf_agents.policies import gaussian_policy
...
class NafAgent(TFAgent):
  def __init__(...):
    ...
    policy = actor_policy.ActorPolicy(...)
    collect_policy = gaussian_policy.GaussianPolicy(policy,
                                                    scale = noise_factor)
    ...
    super(NafAgent, self).__init__(... collect_policy=collect_policy ...)
   ...
```

A continuació, l'agent ja pot invocar el constructor de la superclasse

```
...
class NafAgent(TFAgent):
  def __init__(...):
    ...
    super(NafAgent, self).__init__(
            time_step_spec=time_step_spec,
            action_spec=action_spec,
            policy=policy,
            collect_policy=collect_policy,
            train_sequence_length=2,
            train_step_counter=train_step_counter,
            debug_summaries=debug_summaries,
            summarize_grads_and_vars=summarize_grads_and_vars
            )
    ...
```

El següent pas és inicialitzar les variables necessàries, com les xarxes neuronals

```
...
class NafAgent(TFAgent):
  def __init__(...):
    ...
    # Randomly initialize normalized Q network Q(x,u|0_Q).
    self._v_network.create_variables()
    self._l_matrix_network.create_variables()
    self._policy_network.create_variables()
    ...
```

També caldrà crear la xarxa *target* (si no ha estat proporcionada) i inicialitzar-la. Les funcions del mòdul *utils.common* de TF-Agents ho faciliten.

```
from tf_agents.utils import common
...
class NafAgent(TFAgent):
  def __init__(...):
    ...
    # Create the target network Q' (only the target_v_network is needed)
    ...
    self._target_v_network = common.maybe_copy_target_network_with_checks(
            self._v_network,
            _target_v_network,
            'TargetVNetwork')

    # Initialize target network Q' with weight 0__Q' <- 0_Q
    common.soft_variables_update(self._v_network.variables,
                                 self._target_v_network.variables,
                                 tau=1.0)
   ...
```

La funció *maybe_copy_target_network_with_checks* genera una còpia de la xarxa passada com a primer paràmetre si el segon no està definit o bé, si ho està, comprova que llurs especificacions són compatibles.

La funció *soft_variables_update* actualitza les variables de la segona xarxa en un factor d'actualització *tau* respecte a les de la primera xarxa. En aquest cas, al fixar *tau* a 1.0, les variables de la xarxa *target* prendran els valors íntegres de V.

Finalment, seguint basant-se en el codi equivalent de l'agent DQN de TF-Agents, el constructor defineix la funció d'actualització i periodicitat per a la xarxa *target*.

```
class NafAgent(TFAgent):
  def __init__(...):
    ...
    self._update_target = self._get_target_updater(
                                          tau = self._target_update_tau,
                                          period = self._target_update_period)
    ...

  def _get_target_updater(self, tau=1.0, period=1):
    """Attribution:
       This method has been adapted from:
         https://github.com/tensorflow/agents/blob/master/tf_agents/
         agents/dqn/dqn_agent.py#L359
       Copyright 2020 The TF-Agents Authors.
       Licensed under the Apache License, Version 2.0.
```

```
    """
    with tf.name_scope('update_targets'):
      def update():
        # Only the v_network is used to compute the target values
        return common.soft_variables_update(self._v_network.variables,
                                            self._target_v_network.variables,
                                            tau=tau,
                                            tau_non_trainable=1.0)

      return common.Periodically(update, period, 'update_targets')
```

La funció *Periodically*, del mòdul *common* de TF-Agents, és un wrapper per a la funció *update*, de manera que només s'executa cada cert número d'iteracions definit pel paràmetre *period* (en aquest cas, *target_update_period*).

El mètode *_train*

El mètode *_train* és el mètode principal de l'agent NAF ja que és el que executa l'aprenentatge a partir de les mostres d'experiències viscudes per l'agent.

Per a la implementació d'aquest ha estat de gran utilitat l'anàlisi del mateix mètode en els diferents agents proporcionats per la biblioteca TF-Agents. El mètode implementat és una adaptació del mètode anàleg de l'agent SAC.

```
def _train(self, experience, weights=None):
  """Attribution:
     This method has been adapted from the _train method of the SacAgent class
     from the TF-Agents lib:
       https://github.com/tensorflow/agents/blob/v0.6.0/tf_agents/agents/sac/
       sac_agent.py
     Copyright 2020 The TF-Agents Authors.
     Licensed under the Apache License, Version 2.0.
  """

  # Get the batches in the form of tensors for each concept
  time_steps, action_steps, next_time_steps =
    (trajectory.experience_to_transitions(experience, squeeze_time_dim=True))

  actions = action_steps.action

  # Get the variables to train, deduplicating the shared ones
  variables_to_train = list(
    common.object_identity.ObjectIdentitySet(
      self._v_network.trainable_variables +
      self._l_matrix_network.trainable_variables +
      self._policy_network.trainable_variables))

  # Record the gradient while calculating the loss
  with tf.GradientTape(persistent=True) as tape:
    # Ensure the gradient tape is watching the variables to train
    tape.watch(variables_to_train)
    # Compute the loss L
    loss_info = self._loss(time_steps,
                           actions,
                           next_time_steps,
                           td_errors_loss_fn=self._td_errors_loss_fn,
                           tape=tape,
                           gamma=self._gamma,
                           training=True)
```

```
  # Get the gradients
  grads = tape.gradient(loss_info.loss, variables_to_train)

  # Pack gradients and variables together
  grads_and_vars = list(zip(grads, variables_to_train))
  ...

  # Apply the gradients with the optimizer function
  self._optimizer.apply_gradients(grads_and_vars)

  # Increase the train step counter
  self.train_step_counter.assign_add(1)

  # Update the target network: _0_Q' <- tau*0_Q + (1 - tau)*0_Q'
  self._update_target()

  return loss_info
```

Com es pot observar, el mètode inicialment prepara les dades rebudes en el format necessari per al seu tractament, s'assegura que s'estan observant les variables a optimitzar, i genera un gradient a partir del càlcul de la funció de pèrdua (*loss* o *L*). Per fer-ho utilitza un tipus d'objecte especial anomenat *Gradient Tape* que emmagatzema traces per a les variables durant l'execució del mètode *_loss*. A continuació, el *Gradient Tape* recorre les traces en sentit invers i calcula les components del gradient relatives a cada variable (mètode *gradient*). Obtingut aquest gradient, s'utilitza la funció d'optimització (en aquest TFM s'utilitza *Adam*) per optimitzar les variables entrenables respecte al gradient generat. Finalment, s'incrementa el comptador d'iteracions d'entrenament efectuades i s'actualitza la xarxa *target* d'acord a la funció que retorna el mètode *_update_target* definit en la construcció de l'objecte.

El mètode *_loss*

El mètode _loss és el mètode més complex de l'agent NAF. Permet calcular la diferència temporal entre instants d'una trajectòria i calcular la pèrdua (*loss* o *L*) respecte a la predicció actual de l'agent mitjançant la seva xarxa *target*.

La complexitat del mètode rau en el fet que ha de ser executable tant en el mode eager com en el mode graph de TensorFlow, i això no ha estat senzill, ja que requereix de mètodes paral·lelitzables i, quan hi ha sentències condicionals pel mig, no és fàcil. Per gestionar-ho, s'han utilitzat funcions de TensorFlow que garanteixen aquestes propietats.

Per entendre el funcionament d'aquest mètode s'ha dividit en diferents blocs, corresponents als comentaris del codi que es mostra a continuació.

```
def _loss(self,
          first_time_step: time_step.TimeStep,
          action: policy_step.ActionType,
          next_time_step: time_step.TimeStep,
          td_errors_loss_fn: types.LossFn,
          tape: tf.GradientTape,
          gamma: types.Float = 1.0,
          training: bool = False) -> types.Tensor:
```

```
  with tf.name_scope('loss'):

    with tape.stop_recording():
      # Filter invalid transitions
      ...
      # Set y_i = r_i + gamma*V'(x'_i|0_Q')
      ...
      # Prepare data for Q computation
      ...

    # compute Q(x_i,u_i|0_Q)
    ...
    # calculate loss: L = (1/N)*Sum_i[(y_i - Q(x_i,u_i|0_Q))^2]
    ...
    return LossInfo((avg_loss),())
```

El name_scope facilita l'agrupació de les mètriques generades per a la seva posterior visualització a TensorBoard. D'altra banda, per millorar l'eficiència de l'aprenentatge, totes les operacions de filtratge o de càlcul de *targets* s'han omès de les traces que genera el Gradient Tape, ja que no són rellevants de cara al càlcul del gradient. Sí que és rellevant per a aquest càlcul, la predicció dels valors Q per a l'estat observat i el càlcul de la diferència quadràtica (pèrdua, *loss* o *L*).

El retorn del mètode ha de ser una tupla on el primer element és un tensor que indica el valor de pèrdua obtingut (en aquest cas. la mitjana aritmètica).

Per tant, el primer pas és filtrar i treure del conjunt totes aquelles transicions que no són vàlides per a l'aprenentatge, és a dir, les que tenen com a observació un estat final, o que el següent estat és un estat inicial.

Les transicions vàlides són aquelles en què el primer estat és de tipus MID o FIRST, i el segon estat és de tipus MID o LAST.

Quan el segon estat és de tipus LAST es diu que s'està davant d'una transició terminal (o *boundary*); altrament es tractaria d'una transició no terminal.

```
# Filter invalid transitions

valid_transition_mask = tf.logical_and(
                    first_time_step.step_type != time_step.StepType.LAST,
                    next_time_step.step_type != time_step.StepType.FIRST)

terminal_valid_transition_mask = tf.logical_and(
                    next_time_step.step_type == time_step.StepType.LAST,
                    valid_transition_mask)

non_terminal_valid_transition_mask = tf.logical_and(
                    next_time_step.step_type == time_step.StepType.MID,
                    valid_transition_mask)

non_terminal_transition_observation = tf.boolean_mask(
                    first_time_step.observation,
                    non_terminal_valid_transition_mask)

non_terminal_transition_action = tf.boolean_mask(
                    action,
                    non_terminal_valid_transition_mask)
```

```
non_terminal_transition_next_observation = tf.boolean_mask(
                    next_time_step.observation,
                    non_terminal_valid_transition_mask)

non_terminal_transition_reward = tf.boolean_mask(
                    next_time_step.reward,
                    non_terminal_valid_transition_mask)

terminal_transition_observation = tf.boolean_mask(
                    first_time_step.observation,
                    terminal_valid_transition_mask)

terminal_transition_action = tf.boolean_mask(
                    action,
                    terminal_valid_transition_mask)

terminal_transition_reward = tf.boolean_mask(
                    next_time_step.reward,
                    terminal_valid_transition_mask)
```

A continuació, es calculen els *targets* (valors esperats) per a l'aprenentatge mitjançant la xarxa *target*.

```
# Set y_i = r_i + gamma*V'(x'_i|0_Q')

v_values = self._compute_target_v_values(non_terminal_transition_next_observation)

discounted_v_values = tf.scalar_mul(gamma, v_values)

non_terminal_y_i = tf.add(non_terminal_transition_reward, discounted_v_values)

terminal_y_i = terminal_transition_reward

y_i = tf.concat([non_terminal_y_i, terminal_y_i], 0)
```

Acte seguit, es preparen les dades per tal de tenir un llistat amb les transicions ordenades de la mateixa manera que el llistat *y_i*.

```
# Prepare data for Q computation

valid_observations = tf.concat([ non_terminal_transition_observation,
                                 terminal_transition_observation ],
                               0)

valid_actions = tf.concat([ non_terminal_transition_action,
                            terminal_transition_action ],
                          0)
```

Obtingut aquest llistat, es computa, ara essent observada l'operació per la *Gradient Tape*, els valors Q per a les diferent transicions vàlides proporcionades.

```
# compute Q(x_i,u_i|0_Q)

q_i = self._compute_current_q_values(valid_observations, valid_actions)
```

A continuació, també sota l'observació de la *Gradient Tape*, es calcula el valor de pèrdua entre els valors esperats (*y_i*) i els obtinguts (*q_i*) i la seva mitjana.

```
# calculate loss: L = (1/N)*Sum_i[(y_i - Q(x_i,u_i|0_Q))^2]

td_loss = td_errors_loss_fn(y_i, q_i)
avg_loss = tf.reduce_mean(td_loss)
```

Per facilitar el monitoratge, s'enregistra la traça de la pèrdua en el log per a poder ser observada mitjançant TensorBoard.

```
    losses_dict = {'avg_loss': avg_loss}

    common.summarize_scalar_dict(losses_dict,
                                 step=self.train_step_counter,
                                 name_scope='Losses/')
```

Mètodes per al còmput de les funcions valor

Al llarg de l'explicació de la implantació s'ha mostrat la crida als mètodes que computen les funcions valor, a continuació se'n detalla la implementació.

En primer lloc, s'ha invocat el mètode *_compute_target_v_values*, que crida al mètode *call* de la xarxa neuronal *self._target_v_network*.

```
def _compute_target_v_values(self, observation, training=False):
  v_values = self._target_v_network.call(observation, training)[0]
  return v_values
```

A continuació, s'ha invocat el mètode _compute_current_q_values, que crida a la vegada als mètodes _compute_current_v_values i compute_current_a_values per tal de generar el valor de Q tot sumant els valors obtinguts per a V i A.

```
def _compute_current_q_values(self, observation, action, training=False):
  # Q(x,u|0_Q) = A(x,u|0_A) + V(x|0_V)
  v_values = self._compute_current_v_values(observation, training)
  a_values = self._compute_current_a_values(observation, action, training)
  q_values = tf.add(v_values, a_values)
  return q_values
```

El mètode *_compute_current_v_values* crida el mètode *call* de la xarxa neuronal *self._ v_network*.

```
def _compute_current_v_values(self, observation, training=False):
  v_values = self._v_network.call(observation, training)[0]
  return v_values
```

El mètode *_compute_current_a_values* és més complex. En primer lloc calcula el vector diferència entre les accions executades i les que preveu que s'haurien

d'haver executat d'acord a la política d'explotació actual. A aquesta diferència se l'ha anomenada *delta_action*. Atès que la política rep un *time step* com a paràmetre, d'acord a les especificacions de la biblioteca TF-Agents, s'ha hagut de transformar l'observació a un *time step* trivial. Així mateix, s'ha convertit el vector en matriu (mètode *expand_dims*), per facilitar les operacions de transposició i multiplicació posteriors.

Un cop obtinguda *delta_action*, se sol·licita la matriu *P* al mètode *_get_p_matrix*, tot passant-li com a paràmetre les observacions actuals, i es procedeix a calcular els valors avantatge-acció un cop obtinguda la matriu.

```
def _compute_current_a_values(self, observation, action, training=False):
  # A(x,u|0_A) =
  # ((-1/2)(u - policy(x|0_policy))^T)*P(x|0_P)*(u - policy(x|0_policy))
  time_step = time_step.TimeStep(step_type=None,
                                 reward=None,
                                 discount=None,
                                 observation=observation)

  delta_action_transposed = tf.expand_dims(action, axis=1) -
    tf.expand_dims(self._policy.action(time_step).action, axis=1)

  delta_action = tf.linalg.matrix_transpose(delta_action_transposed)

  p_matrix = self._get_p_matrix(observation)
  aux = tf.linalg.matmul(delta_action_transposed, p_matrix)
  aux = tf.linalg.matmul(aux, delta_action)
  a_value = tf.squeeze(tf.math.scalar_mul(-0.5, aux), axis=[-1, -2])
  return a_value
```

El mètode *_get_p_matrix* obté la matriu P a partir de la matriu L, que obté del mètode *_get_l_matrix*, al qual també se li passa com a paràmetre les observacions actuals.

```
def _get_p_matrix(self, state):
  # P(x|0_P) = L(x|0_P)*L(x|0_p)^T
  l_matrix = self._get_l_matrix(state)
  return tf.linalg.matmul(l_matrix,tf.linalg.matrix_transpose(l_matrix))
```

Finalment, el mètode *_get_l_matrix* genera la matriu triangular L a partir dels valors rebuts de la invocació del mètode *call* de la xarxa neuronal *self._l_matrix_network*.

```
def _get_l_matrix(self, state):
  # L(x|0_P) is a lower-triangular matrix whose entries come from a linear
  # output layer of a neural network, with the diagonal terms exponentiated

  l_matrix_terms = self._l_matrix_network.call(state, training=True)[0]
  pre_l_matrix = tfp.math.fill_triangular(l_matrix_terms)
  return tf.linalg.set_diag(pre_l_matrix,
                            tf.math.exp(tf.linalg.diag_part(pre_l_matrix)))
```

7.4 Implementació del mòdul experiments

La classe principal d'aquest mòdul és l'*OffPolicyTimeStepBasedExperiment*. Aquesta classe és una adaptació del codi agents.sac.examples.v2.train_eval.py de la biblioteca TF-Agents, al qual se li ha donat estructura de classe i s'ha generalitzat per funcionar amb qualsevol tipus d'agent, no només SAC. Per manca d'espai, es mostrarà únicament en aquesta memòria l'obtenció de l'agent per part de la superclasse i de les subclasses, i es deixarà la resta de la implementació com a document annex per qui vulgui consultar-la.

```
class OffPolicyTimeStepBasedExperiment(object):
  """A simple train and eval class for Off-Policy TimeStep based Agents."""
  def __init__(...):
    ...

  def launch(self):
    ...
    tf_agent = self._get_the_agent(self._get_the_agent(time_step_spec,
                                    observation_spec,
                                    action_spec,
                                    global_step,
                                    self.debug_summaries,
                                    self.summarize_grads_and_vars)
    tf_agent.initialize()
    ...
```

```
class RndExperiment(OffPolicyTimeStepBasedExperiment):
  """A simple train and eval class for a Random Policy Agent."""
  def __init__(...):
   …

 def _get_the_agent(...) -> TFAgent:
   return random_agent.RandomAgent(
                               time_step_spec,
                               action_spec,
                               train_step_counter=global_step,
                               debug_summaries=debug_summaries,
                               summarize_grads_and_vars=summarize_grads_and_vars,
                               num_outer_dims=2)
```

```
class SacExperiment(OffPolicyTimeStepBasedExperiment):
  """A simple train and eval class for a SAC agent."""
  def __init__(...):
    ...

  def _get_the_agent(...) -> TFAgent:

  actor_net = actor_distribution_network.ActorDistributionNetwork(...)
  critic_net = critic_network.CriticNetwork(...)

  tf_agent = sac_agent.SacAgent(time_step_spec,
                                action_spec,
                                actor_network=actor_net,
                                critic_network=critic_net,
                                ...)

  return tf_agent:
```

```
class NafExperiment(OffPolicyTimeStepBasedExperiment):
  """A simple train and eval class for a NAF Agent."""
  def __init__(...):
    ...

  def _get_the_agent(self, time_step_spec, observation_spec, action_spec, ...)
                      -> TFAgent:

        # Q Function Networks

        # Shared Preprocessing Network (Optional)

        preprocessing_network = encoding_network.EncodingNetwork(
            observation_spec,
            ...)

        preprocessing_network.create_variables()

        # V-value Network
        v_value_network = CompositeValueNetwork(
            observation_spec,
            ...)

        v_value_network.set_shared_network (preprocessing_network)

        # L-Matrix Network
        l_matrix_network = CompositeLNetwork(
            observation_spec,
            action_spec,
            ...)

        l_matrix_network.set_shared_network (preprocessing_network)

        policy_network = CompositeActorDistributionNetwork(
            observation_spec,
            action_spec,
            ...)

        if self.policy_network_uses_shared_preprocessing_network:
            policy_network.set_shared_network(preprocessing_network)

        tf_agent = NafAgent(time_step_spec,
                            action_spec,
                            v_network=v_value_network,
                            l_matrix_network=l_matrix_network,
                            policy_network=policy_network,
                            ...)

        return tf_agent
```

Com es pot observar, per a la creació de les xarxes de l'agent NAF s'ha fet ús d'un tipus especial de xarxes amb prefix *Composite*. Aquestes xarxes han estat dissenyades per l'estudiant per en aquest TFM i permeten compartir una xarxa neuronal ja inicialitzada (és a dir, per a la qual s'ha invocat el mètode *create_variables*). Per manca d'espai en la memòria, no es comentarà la seva implementació, tot i que se n'annexarà en codi en els documents annexos.

7.5 Creació i llançament d'experiments

La creació i llançament d'experiments s'ha dut a terme des del mòdul main.py a l'arrel del projecte.

En concret, els experiments s'instancien, es repliquen si s'escau (invocant el mètode *copy*), es parametritzen adientment i es posen en un llistat per tal de ser llançats (mètode launch).

Per llençar els experiments en paral·lel s'utilitza la biblioteca *multiprocessing*.

```
import multiprocessing
...

naf_base_experiment = NafExperiment(...)

naf_experiment_1 = naf_base_experiment.copy()
naf_experiment_1.name = "naf_experiment_1"
naf_experiment_1.train_steps_per_iteration=10

naf_experiment_2 = naf_base_experiment.copy()
naf_experiment_2.name = "naf_experiment_1"
naf_experiment_2.train_steps_per_iteration=20

experiment_list = [naf_experiment_1, naf_experiment_2]

if(len(experiment_list) == 1): # Single process execution
  experiment_list[0].launch()

else:                          # Multiple processes execution
  processes_list = list()

  if __name__ == "__main__":

    for experiment in experiment_list:
      process = multiprocessing.Process(target = launch_experiment,
                                        args=[experiment])
      processes_list.append(process)
      process.start()
      print("Process \"" + str(experiment.name) +
            "\" started with pid = " + str(process.pid))
```

7.6 Requisits no funcionals

Els requisits no funcionals de poder desar l'estat de l'execució en qualsevol moment i reprendre'l més endavant o el de poder renderitzar l'entorn durant l'entrenament i anular aquesta renderització a voluntat no s'han acabat implementant. Malgrat es va iniciar el seu desenvolupament i proves mitjançant la biblioteca *pynput*, el *listener* capturava tots els esdeveniments de teclat del sistema, no només els de la consola des d'on s'estava executant l'aplicació i, per tant, resultava poc pràctic. A més a més, el programa trencava sovint el seu flux d'execució i calia reprendre'l de nou. Per manca de temps no s'ha pogut investigar com solucionar aquest problema.

8 Resultats

8.1 Proves amb l'agent SAC i l'*ActionRepeatHistoryWrapper*

L'*ActionRepeatHistoryWrapper* implementat, s'ha provat en els entorns BipedalWalker-v3 i BipedalWalkerHardcore-v3 amb l'agent SAC de TF-Agents. Mantenint els paràmetres constants per a cada entorn[26], s'han provat de compactar diferent nombre de *time steps* per part del *wrapper*, tot detectant millora en els resultats de l'agent a partir de 3 o més *time steps* compactats (Figura 26). Els experiments s'han pogut replicar diferents cops amb resultats similars, la qual cosa indica que la implementació del *wrapper* ha estat una bona estratègia.

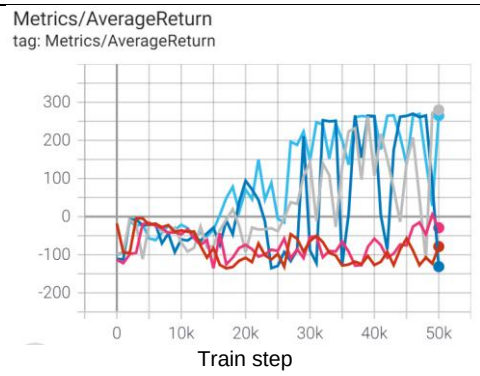
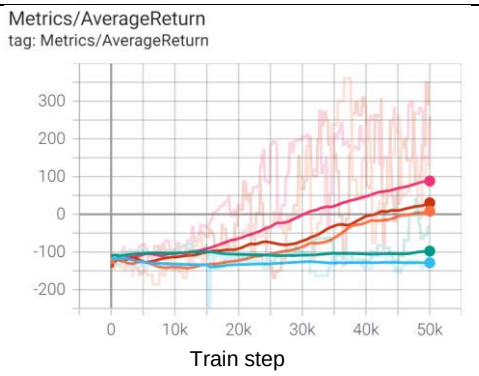
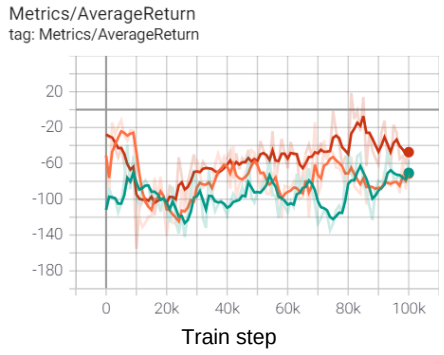
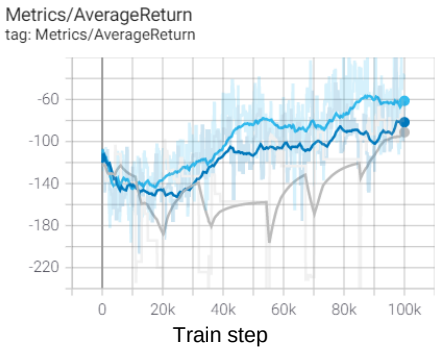
<table>
<tr><th colspan="2">Entorn BipedalWalker-v3</th></tr>
<tr><th>Avaluació</th><th>Entrenament</th></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>Time steps compactats pel wrapper :
1 2 3 4 5</td><td>Time steps compactats pel wrapper :
1 2 3 4 5</td></tr>
<tr><th colspan="2">Entorn BipedalWalkerHardcore-v3</th></tr>
<tr><th>Avaluació</th><th>Entrenament</th></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>Time steps compactats pel wrapper :
1 4 8</td><td>Time steps compactats pel wrapper :
1 4 8</td></tr>
</table>

Figura 26. Resultats de l'execució del wrapper RepeatActionEnvWrapper amb l'agent SAC de TF-Agents[27]

En l'entorn BipedalWalkerHardcore-v3 els resultats han estat menys exitosos. Incrementant la complexitat de les xarxes neuronals i analitzant els vídeos obtinguts dels experiments es detecta una clara millora en les polítiques obtingudes amb compactació de *time steps*. Adjunts a aquesta memòria s'han lliurat també els vídeos generats en els diferents experiments.

[26] Els paràmetres es recullen a l'Annex 12.5: Relació d'experiments proporcionats i paràmetres emprats

[27] S'ha aplicat un procediment de *smoothing* als gràfics per facilitar-ne la interpretació.

8.2 Proves amb l'agent NAF implementat

Els experiments es van iniciar utilitzant el *wrapper* implementat tot compactant 4 *time steps* per cada iteració d'interacció amb l'entorn, ja que era el nombre a partir del qual s'obtenia una millora suficient amb l'agent SAC. Es van provar diferents configuracions i tipus de xarxes neuronals amb i sense xarxa de preprocessament compartida, taxes d'aprenentatge, valors *tau*, capacitats del *replay buffer*, nombre d'iteracions d'entrenament per cada iteració d'interacció amb l'entorn, polítiques d'addició de soroll i llurs paràmetres ($\in$-greedy, soroll gaussià, soroll d'Ornstein-Uhlenbeck, fins i tot combinacions d'aquestes), etc. En cap cas es va aconseguir superar els resultats obtinguts per la política aleatòria.

Finalment, el dia abans del *deadline* per al lliurament de la memòria, compactant 8 *time steps* de l'entorn en comptes de 4 es va començar a obtenir resultats positius per a l'entorn BipedalWalker-v3. Aquests es van poder replicar diversos cops (Figura 27), descartant així que el resultat fos efecte de l'atzar. En conseqüència, s'ha pogut provar finalment que l'algorisme implementat funciona, malgrat la complexitat del problema és força elevada per a l'agent. De fet, no hi ha hagut èxit amb l'entorn BipedalWalkerHardcore-v3.

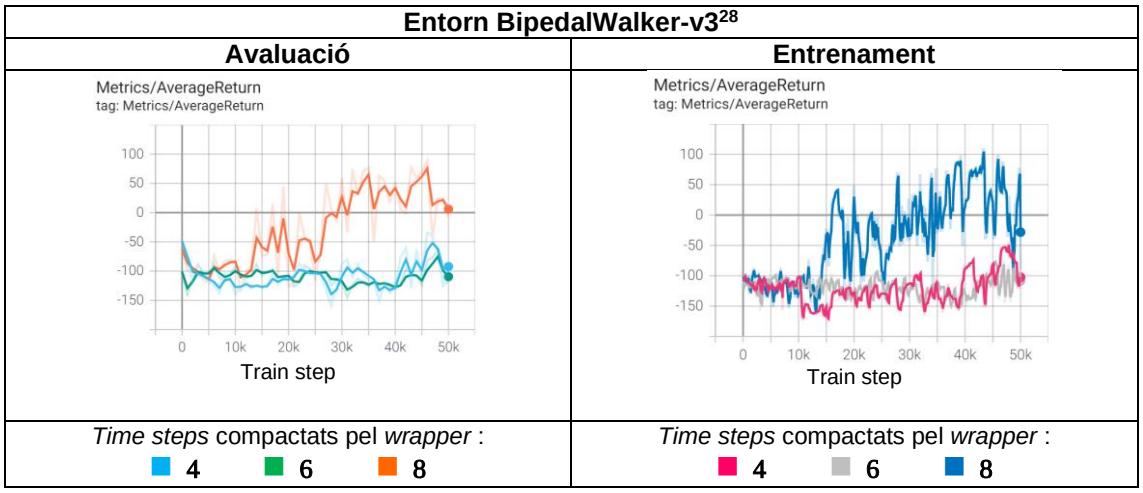


Figura 27. Resultats de l'execució de l'entorn BipedalWalker2D-v3 dins el wrapper RepeatActionEnvWrapper

8.3 Mode *graph* de TensorFlow vs mode *eager*

Quan l'agent NAF s'executa en mode *graph* l'eficiència computacional és molt superior, gairebé d'un ordre de magnitud (Figura 28).

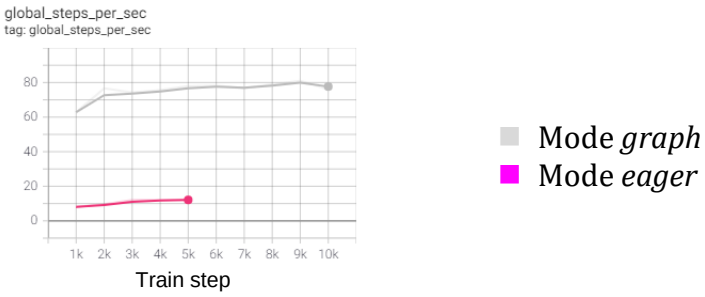


Figura 28. Velocitats d'execució de l'agent NAF en mode graph i mode eager per al mateix problema i configuració

[28] S'ha aplicat un procediment de *smoothing* als gràfics per facilitar-ne la interpretació.

9 Conclusions

9.1 Quant al treball desenvolupat

- El *Deep Reinforcement Learning* (DRL) és una disciplina complexa, que ha avançat molt en els darrers anys i que ofereix solucions molt diverses als diferents problemes que intenta resoldre.

- Els problemes que utilitzen espai d'accions continus són força més complexes de resoldre que els que utilitzen espais d'accions discrets.

- La implementació d'un agent DRL no és una tasca trivial, hi ha molts factors a tenir presents i els paràmetres que funcionen en un tipus d'agent poden no ser adequats o suficients en altres tipus d'agents.

- El temps d'execució quan s'utilitza el mode de grafs de la biblioteca TensorFlow disminueix considerablement en relació al mode *eager*; la programació, però, esdevé més complexa, així com la depuració del codi.

- TF-Agents és una biblioteca molt potent, però encara es troba en desenvolupament i manca d'alguns elements importants, com per exemple de la possibilitat de crear diferents xarxes amb capes compartides (potser per aquest motiu no té cap agent *Dueling DQN* implementat). Malgrat en aquest TFM s'ha implementat una alternativa per poder fer ús d'aquests tipus de xarxes, la solució no segueix la filosofia original de la biblioteca TF-Agents.

- L'agent SAC és molt més eficient aprenent que no pas l'agent NAF per al problema en qüestió, al menys quan NAF no utilitza acceleració per model.

- La creació del *wrapper* per compactar diferents *time steps* com si fossin un de sol ha incrementat l'eficiència de l'agent amb l'entorn, ja que li permet tenir una percepció de la dimensió temporal. Aquest mètode, però té els seus límits, ja que com més *time steps* es compacten, menor velocitat de reacció tindrà l'agent. Conceptualment, la compactació de *time steps* porta el problema a aproximar-se més a la propietat de Markov, ja que permet a l'agent predir el comportament de l'entorn i, per tant, la seva interpretació del que està succeint no depèn tant de les accions del passat.

- L'ús de la biblioteca TF-Agents, tot i que ha retardat l'obtenció d'una solució, ha permès aprofundir molt en la comprensió del DRL.

9.2 Quant als objectius plantejats

- Quant a l'objectiu general:

    - L'objectiu general del projecte s'ha assolit satisfactòriament, obtenint un agent operatiu que aproxima la solució al problema, malgrat quedaria encara molta feina per fer per tal d'optimitzar-lo.

- Quant als objectius específics:
    - Els objectius d'analitzar l'estat de l'art del problema en qüestió i dels paradigmes del Reinforcement Learning i del Deep Reinforcement Learning s'han pogut acomplir satisfactòriament i han resultat molt productius pel que fa als coneixements adquirits per part de l'estudiant.
    - Els objectius relacionats amb l'anàlisi i ús de la plataforma OpenAI Gym també s'han acomplert satisfactòriament, havent-la emprat com a base per al desenvolupament del producte.
    - Quant als objectius relacionats amb el disseny de la solució, s'han escollit algorismes molt eficients per al treball en una màquina domèstica i s'han pogut parametritzar i obtenir resultats en pocs minuts (i en algun cas, poques hores). Tanmateix, no s'han pogut implementar els requisits no funcionals.
    - L'objectiu d'integrar de forma holística diferents paradigmes en un sol agent i els objectius derivats d'aquest no s'han pogut assolir, ja que la dilatació de l'estudi de l'estat de l'art i la dificultat de parametrització de l'agent NAF han esgotat el temps del projecte.
    - Finalment, pels mateixos motius que en el cas de l'objectiu anterior, l'objectiu d'escalar la solució a un entorn 3D no s'ha pogut abordar i, per tant, no s'ha pogut acomplir.

9.3 Quant a la planificació

- La planificació s'ha pogut seguir correctament, aplicant les contingències previstes en el pla de riscos quan hi ha hagut desviacions. En concret, s'ha d'eliminar de l'abast la implementació de la solució 3D (canvi proposat en la PAC2) i la integració de metodologies (canvi proposat en la PAC3).

9.4 Línies de treball futur

- Provar d'implementar l'acceleració de NAF mitjançant la planificació amb un model après a partir de l'experiència (Gu, Lillicrap, Sutskever, & Levine, 2016), aquesta millora podria portar finalment a resultats positius.
- Provar d'implementar una funció de predicció del risc de caiguda i afegir-la en un *wrapper* en la política d'exploració, tot analitzant com afecta aquest fet en l'aprenentatge (García & Shafie, 2020).
- Millorar la funcionalitat de les classes CompositeNeuronalNetwork i derivades, de manera que siguin conformes a la filosofia de la biblioteca TF-Agents.
- Posar el codi desenvolupat a disposició de l'equip de TF-Agents per si en volen aprofitar alguna part, contribuint així en la generació i difusió del coneixement.

10 Glossari

RL	*Reinforcement Learning*
DRL	*Deep Reinforcement Learning*
MDP	*Markov Decission Process*
POMDP	*Partialy Observable Markov Decission Process*
FOMDP	*Fully Observable Markov Decission Process*
DP	*Dynamic Programming*
MC	*Monte Carlo*
TD learning	*Time Difference learning*
DQN	*Deep Q-Network*
NAF	*Normalized Advantage Function*
SAC	*Soft Actor Critic*
CAQL	*Continuous Action Q-Learning*

11 Bibliografia[29]

Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (11 / 2017). Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Processing Magazine, 34*(6), 26-38. doi:10.1109/MSP.2017.2743240

Benítez, R., Escudero, G., Kanaan, S., Masip Rodó, D., & Cencerrado Barraqué, A. (2018). *Intel·ligència Artificial Avançada* (Segona ed.). Barcelona: Universitat Oberta de Catalunya.

Beysolow, T., Taweh, & author. (2019). *Applied reinforcement learning with Python : with OpenAI Gym, Tensorflow and Keras.* Apress. Recollit de http://mendeley.csuc.cat/fitxers/448f5e41e4b0241498b0448605f88163

Bogue, R. (3 / 2014). The role of artificial intelligence in robotics. *Industrial Robot: An International Journal, 41*(2), 119-123. doi:10.1108/IR-01-2014-0300

Cearley, D., Jones, N., Smith, D., Burke, B., Chandrasekaran, A., Lu, C. K., & Panetta, K. (2019). Gartner Top 10 Strategic Technology Trends for 2020 - Smarter With Gartner. *Gartner*, 52. Recollit de https://www.gartner.com/smarterwithgartner/gartner-top-10-strategic-technology-trends-for-2020 https://www.gartner.com/smarterwithgartner/gartner-top-10-strategic-technology-trends-for-2020/

Clarisó Viladrosa, R. (2013). *Treball final de màster / Robert Clarisó Viladrosa, José Ramón Rodríguez.* Universitat Oberta de Catalunya,. Recollit de http://discovery.uoc.edu/iii/encore/record/C__Rb1023732__STreball%20final%20de%20m%E0ster%20Robert%20Claris%F3%20Viladrosa__Orighthresult__U__X4?lang=cat

Coumans, E., & Bai, Y. (2020). *PyBullet Quickstart Guide.* Recollit de https://docs.google.com/document/d/10sXEhzFRSnvFcl3XxNGhnD4N2SedqwdAvK3dsihxVUA/edit#heading=h.2ye70wns7io3

Fayek, H. M., Cavedon, L., & Wu, H. R. (2020). Progressive learning: A deep learning framework for continual learning. *Neural Networks, 128*, 345-357. doi:https://doi.org/10.1016/j.neunet.2020.05.011

García, J., & Shafie, D. (2020). Teaching a humanoid robot to walk faster through Safe Reinforcement Learning. *Engineering Applications of Artificial Intelligence, 88*, 103360. doi:https://doi.org/10.1016/j.engappai.2019.103360

Google Research. (09 / 11 / 2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. Preliminary White Paper.* Consultat el 14 / 12 / 2020, a TensorFlow White Papers:

[29] Per a l'elaboració de la bibliografia s'han tingut en consideració els següents elements:

- S'ha fet ús de les opcions de gestió bibliogràfica que ofereix el programari Microsoft Word, amb el format de cita APA.
- Per a l'ordre de cita dels/les autors/es s'ha utilitzat aquell amb què se citen en la publicació.
- Per a l'obtenció i gestió de les cites s'ha utilitzat el programari *Mendeley Reference Manager*.

https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45166.pdf

Gu, S., Lillicrap, T., Sutskever, I., & Levine, S. (2016). Continuous Deep Q-Learning with Model-based Acceleration. *arXiv:1603.00748*.

Gu, S., Lillicrap, T., Sutskever, I., & Levine, S. (02 / 03 / 2016). Continuous Deep Q-Learning with Model-based Acceleration. *arXiv:1603.00748*.

Gura, T. (2013). Citizen science: Amateur experts. *Nature, 496*(7444), 259-261.

Ha, D. (2018). Reinforcement Learning for Improving Agent Design. *arXiv:1810.03779*.

Haarnoja, T., Tang, H., Abbeel, P., & Levine, S. (2017). Reinforcement Learning with Deep Energy-Based Policies. *arXiv:1702.08165*.

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *arXiv:1801.01290*. Recollit de https://arxiv.org/abs/1801.01290

Holt, S. (2020). *Value and Policy based agents.* Consultat el 14 / 12 / 2020, a Path to Pioneer: https://pathtopioneer.com/blog/2020/07/rl-4

Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., . . . Levine, S. (2018). QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation. *arXiv:1806.10293*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., . . . Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv:1509.02971*.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (12 / 2013). Playing Atari with Deep Reinforcement Learning. Recollit de https://arxiv.org/abs/1312.5602

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., . . . Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529-533. doi:10.1038/nature14236

Mnih, V., Puigdomènech Badia, A., Mirza, M., Graves, A., . Lillicrap, T. P., Harley, T., . . . Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. *arXiv:1602.01783*.

Nachum, O., Norouzi, M., Xu, K., & Schuurmans, D. (2017). Bridging the Gap Between Value and Policy Based Reinforcement Learning. *arXiv:1702.08892*.

Nian, R., Liu, J., & Huang, B. (2020). A review On reinforcement learning: Introduction and applications in industrial process control. *Computers & Chemical Engineering, 139*, 106886. doi:https://doi.org/10.1016/j.compchemeng.2020.106886

Palanisamy, P., & Palanisamy, P. (sense data). Hands-On Intelligent Agents with OpenAI Gym: Your Guide to Developing AI Agents Using Deep Reinforcement Learning. *Hands-On Intelligent Agents with OpenAI Gym: Your Guide to Developing AI Agents Using Deep Reinforcement Learning*. Recollit de

http://search.ebscohost.com/login.aspx?direct=true&db=&AN=&site=eds-live

PMI, P. M. (2017). *A guide to the project management body of knowledge (PMBOK guide).* Project Management Institute, Inc. Recollit de https://discovery.uoc.edu/iii/encore/record/C__Rb1065164__Spmbok__Orightresult__U__X4?lang=cat

Ryu, M., Chow, Y., Anderson, R., Tjandraatmadja, C., & Boutilier, C. (2019). CAQL: Continuous Action Q-Learning. *arXiv:1909.12397*.

Salimans, T., Ho, J., Chen, X., Sidor, S., & Sutskever, I. (2017). Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *arXiv:1703.03864*.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2017). Trust Region Policy Optimization. *arXiv:1502.05477*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv:1707.06347*.

Serrano-Cuevas, J., Morales, E. F., & Hernández-Leal, P. (8 / 2020). Safe reinforcement learning using risk mapping by similarity. *Adaptive Behavior, 28*(4), 213-224. doi:10.1177/1059712319859650

Shin, S. J., Cho, C. L., Jeon, H. S., Yoon, S. H., & Kim, T. Y. (2019). A Survey on Deep Reinforcement Learning Libraries. *Electronics and Telecommunications Trends (전자통신동향분석), 34*(6), 87-89. doi:https://doi.org/10.22648/ETRI.2019.J.340608

Silva, P., Santos, C. P., Matos, V., & Costa, L. (2014). Automatic generation of biped locomotion controllers using genetic programming. *Robotics and Autonomous Systems, 62*(10), 1531-1548. doi:https://doi.org/10.1016/j.robot.2014.05.008

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014). Deterministic policy gradient algorithms. *Proceedings of the 31 st International Conference on Machine Learning*, 387-395.

Simonini, T. (2020). *On Choosing a Deep Reinforcement Learning Library.* Consultat el 14 / 12 / 2020, a data iku: https://blog.dataiku.com/on-choosing-a-deep-reinforcement-learning-library

Sutton, R. S. (1990). Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. *Proceedings of the Seventh International Conference*, 216-224.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning, Second Edition : An Introduction* (Segona ed.). Cambridge, Massachussets: MIT Press.

van Hasselt, H. (2018). *Reinforcement Learning Course | DeepMind & UCL.* Consultat el 12 / 11 / 2020, a https://www.youtube.com/playlist?list=PLqYmG7hTraZBKeNJ-JE_eyJHZ7XgBoAyb

van Hasselt, H., Guez, A., & Silver, D. (2015). Deep Reinforcement Learning with Double Q-learning. *arXiv:1509.06461*.

van Seijen, H., Fatemi, M., Romoff, J., Laroche, R., Barnes, T., & Tsang, J. (2017). Hybrid Reward Architecture for Reinforcement Learning. *arXiv:1706.04208*.

Wang, R., Lehman, J., Clune, J., & Stanley, K. O. (21 / February / 2019). Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions. (arXiv:1901.01753 [cs.NE]). Recollit de https://arxiv.org/abs/1901.01753

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., & de Freitas, N. (2016). Dueling Network Architectures for Deep Reinforcement Learning. *arXiv:1511.06581*.

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

Xi, A., & Chen, C. (8 / 2020). Stability Control of a Biped Robot on a Dynamic Platform Based on Hybrid Reinforcement Learning. *Sensors, 20*(16), 4468. doi:10.3390/s20164468

Yu, W., Kumar, V. C., Turk, G., & Liu, C. K. (11 / 2019). Sim-to-Real Transfer for Biped Locomotion. (p. 3503-3510). IEEE. doi:10.1109/IROS40897.2019.8968053

Zhang, H., & Yu, T. (2020). Taxonomy of Reinforcement Learning Algorithms. A H. Dong, Z. Ding, & S. Zhang (Ed.), *Deep Reinforcement Learning.* Singapore: Springer.

12 Annexos

12.1 Codi font de la classe OffPolicyTimeStepBasedExperiment

```
# coding=utf-8
# Copyright 2020 Rafael Jesús Castaño Ribes.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

"""Train and Eval class for off-policy TD learning agents feeded with batches
of experiences from a replay buffer.
Attribution:
  This class is an adaptation of the module train_eval.py from the SAC Agent
examples from the TF-Agents library:

https://github.com/tensorflow/agents/blob/v0.6.0/tf_agents/agents/sac/examples/
v2/train_eval.py
  Copyrighted 2020 by the TF-Agents Authors and licensed under the Apache
License, Version 2.0.
Changes added to the original code:
  The code has been given a class format and has been generalized to work with
any off-policy TD based TF-Agent, wich will have to be provided by
  subclasses via the method _get_the_agent().
  Moreover, a method called create_video has been added allowing the class to
generate a video by playing the obtained exploitation policy on the
environment. This method has been adapted from the example code provided by
  Holt, S. (2020). Value and Policy based agents:
https://pathtopioneer.com/blog/2020/07/rl-4, licensed under a Creative Commons
CC BY-NC-SA 4.0 license.
"""

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import os
import time

from absl import logging

from six.moves import range
import tensorflow as tf  # pylint: disable=g-explicit-tensorflow-version-import

from tf_agents.agents.tf_agent import TFAgent
from tf_agents.agents.random import random_agent
from tf_agents.drivers import dynamic_step_driver

from tf_agents.environments import suite_gym
from environments.wrappers import ActionRepeatHistoryWrapper
from tf_agents.environments import tf_py_environment

from tf_agents.eval import metric_utils
```

```
from tf_agents.metrics import tf_metrics

from tf_agents.policies import greedy_policy
from tf_agents.policies import random_tf_policy

from tf_agents.replay_buffers import tf_uniform_replay_buffer
from tf_agents.utils import common

import imageio

class OffPolicyTimeStepBasedExperiment(object):
  """A simple train and eval class for Off-Policy TimeStep based Agents"""
  def __init__(self,
               root_dir = "",
               num_eval_episodes = 10,
               summary_interval = 1000,
               env_name = "BipedalWalker-v3",
               env_action_repeat_times = 4,
               debug_summaries=False,
               summarize_grads_and_vars=False,
               replay_buffer_capacity=1000000,
               initial_collect_steps=10000,
               collect_steps_per_iteration=1,
               use_tf_functions=True,
               sample_batch_size = 256,
               num_iterations=3000000,
               train_steps_per_iteration=1,
               log_interval=1000,
               eval_interval=10000,
               train_checkpoint_interval=50000,
               policy_checkpoint_interval=50000,
               replay_buffer_checkpoint_interval=50000,
               name="default_naf_experiment"
               ):

    self.root_dir = root_dir
    self.num_eval_episodes = num_eval_episodes
    self.summary_interval = summary_interval
    self.env_name = env_name
    self.env_action_repeat_times = env_action_repeat_times
    self.debug_summaries = debug_summaries
    self.summarize_grads_and_vars = summarize_grads_and_vars
    self.replay_buffer_capacity = replay_buffer_capacity
    self.initial_collect_steps = initial_collect_steps
    self.collect_steps_per_iteration = collect_steps_per_iteration
    self.use_tf_functions = use_tf_functions
    self.sample_batch_size = sample_batch_size
    self.num_iterations = num_iterations
    self.train_steps_per_iteration = train_steps_per_iteration
    self.log_interval = log_interval
    self.eval_interval = eval_interval
    self.train_checkpoint_interval = train_checkpoint_interval
    self.policy_checkpoint_interval = policy_checkpoint_interval
    self.replay_buffer_checkpoint_interval = replay_buffer_checkpoint_interval
    self.name = name

  def copy(self):
    return self._copy()

  def _copy(self):
    return OffPolicyTimeStepBasedExperiment(
            root_dir = self.root_dir,
```

```
            num_eval_episodes = self.num_eval_episodes,
            summary_interval = self.summary_interval,
            env_name = self.env_name,
            env_action_repeat_times = self.env_action_repeat_times,
            debug_summaries = self.debug_summaries,
            summarize_grads_and_vars = self.summarize_grads_and_vars,
            replay_buffer_capacity = self.replay_buffer_capacity,
            initial_collect_steps = self.initial_collect_steps,
            collect_steps_per_iteration = self.collect_steps_per_iteration,
            use_tf_functions = self.use_tf_functions,
            sample_batch_size = self.sample_batch_size,
            num_iterations = self.num_iterations,
            train_steps_per_iteration = self.train_steps_per_iteration,
            log_interval = self.log_interval,
            eval_interval = self.eval_interval,
            train_checkpoint_interval = self.train_checkpoint_interval,
            policy_checkpoint_interval = self.policy_checkpoint_interval,
            replay_buffer_checkpoint_interval =
self.replay_buffer_checkpoint_interval,
            name = self.name
            )

  def launch(self):

    tf.compat.v1.enable_v2_behavior()
    logging.set_verbosity(logging.INFO)

    base_dir = os.path.join(self.root_dir, self.name)

    train_dir = os.path.join(base_dir, 'train')
    eval_dir = os.path.join(base_dir, 'eval')

    train_summary_writer = tf.compat.v2.summary.create_file_writer(train_dir)
    train_summary_writer.set_as_default()

    eval_summary_writer = tf.compat.v2.summary.create_file_writer(eval_dir)

    eval_metrics = [
      tf_metrics.AverageReturnMetric(buffer_size=self.num_eval_episodes),
      tf_metrics.AverageEpisodeLengthMetric(buffer_size=self.num_eval_episodes)
      ]

    global_step = tf.compat.v1.train.get_or_create_global_step()

    with tf.compat.v2.summary.record_if(
      lambda: tf.math.equal(global_step % self.summary_interval, 0)):

    py_env = ActionRepeatHistoryWrapper(suite_gym.load(self.env_name),
                                        times=self.env_action_repeat_times)

    tf_env = tf_py_environment.TFPyEnvironment(py_env)

    eval_py_env = ActionRepeatHistoryWrapper(suite_gym.load(self.env_name),
                                        times=self.env_action_repeat_times)

    eval_tf_env = tf_py_environment.TFPyEnvironment(py_env)

    time_step_spec = tf_env.time_step_spec()
    observation_spec = time_step_spec.observation
    action_spec = tf_env.action_spec()

    # Get the agent; Subclasses must override the _get_the_agent method
```

```
    tf_agent = self._get_the_agent(time_step_spec,
                                   observation_spec,
                                   action_spec,
                                   global_step,
                                   self.debug_summaries,
                                   self.summarize_grads_and_vars)

    tf_agent.initialize()

    # Make the replay buffer.
    replay_buffer = tf_uniform_replay_buffer.TFUniformReplayBuffer(
                data_spec=tf_agent.collect_data_spec,
                batch_size=tf_env.batch_size,
                max_length=self.replay_buffer_capacity)

    replay_observer = [replay_buffer.add_batch]

    train_metrics = [tf_metrics.NumberOfEpisodes(),
                     tf_metrics.EnvironmentSteps(),
                     tf_metrics.AverageReturnMetric(
                     buffer_size = self.num_eval_episodes,
                                   batch_size=tf_env.batch_size),
                     tf_metrics.AverageEpisodeLengthMetric(
                                   buffer_size=self.num_eval_episodes,
                                   batch_size=tf_env.batch_size),
                     ]

    eval_policy = greedy_policy.GreedyPolicy(tf_agent.policy) if not
          isinstance(tf_agent, random_agent.RandomAgent) else tf_agent.policy

    initial_collect_policy = random_tf_policy.RandomTFPolicy(time_step_spec,
                                                             action_spec)

    collect_policy = tf_agent.collect_policy

    train_checkpointer = common.Checkpointer(
            ckpt_dir=train_dir,
            agent=tf_agent,
            global_step=global_step,
            metrics=metric_utils.MetricsGroup(train_metrics, 'train_metrics'))

    policy_checkpointer = common.Checkpointer(
            ckpt_dir=os.path.join(train_dir, 'policy'),
            policy=eval_policy,
            global_step=global_step)

    replay_buffer_checkpointer = common.Checkpointer(
            ckpt_dir=os.path.join(train_dir, 'replay_buffer'),
            max_to_keep=1,
            replay_buffer=replay_buffer)

    train_checkpointer.initialize_or_restore()
    replay_buffer_checkpointer.initialize_or_restore()

    initial_collect_driver = dynamic_step_driver.DynamicStepDriver(
            tf_env,
            initial_collect_policy,
            observers=replay_observer + train_metrics,
            num_steps=self.initial_collect_steps)

    collect_driver = dynamic_step_driver.DynamicStepDriver(
            tf_env,
```

```
            collect_policy,
            observers=replay_observer + train_metrics,
            num_steps=self.collect_steps_per_iteration)

    if self.use_tf_functions:
      initial_collect_driver.run = common.function(initial_collect_driver.run)
      collect_driver.run = common.function(collect_driver.run)
      tf_agent.train = common.function(tf_agent.train)

    if replay_buffer.num_frames() == 0:
      # Collect initial replay data.
      logging.info('Initializing replay buffer by collecting experience for %d
                   steps with a random policy.', self.initial_collect_steps)

      initial_collect_driver.run()

    results = metric_utils.eager_compute(eval_metrics,
                                         eval_tf_env,
                                         eval_policy,
                                         num_episodes=self.num_eval_episodes,
                                         train_step=global_step,
                                         summary_writer=eval_summary_writer,
                                         summary_prefix='Metrics',)

    metric_utils.log_metrics(eval_metrics)

    time_step = None

    policy_state = collect_policy.get_initial_state(tf_env.batch_size)

    timed_at_step = global_step.numpy()

    time_acc = 0

    # Prepare replay buffer as dataset with invalid transitions filtered.
    def _filter_invalid_transition(trajectories, unused_arg1):
      return ~trajectories.is_boundary()[0]

    dataset = replay_buffer.as_dataset(
                sample_batch_size = self.sample_batch_size,
                num_steps = 2
                ).unbatch().filter(
                    _filter_invalid_transition
                    ).batch(
                        self.sample_batch_size
                        ).prefetch(5)

    # Dataset generates trajectories with shape [Bx2x...]
    iterator = iter(dataset)

    def train_step():
      experience, _ = next(iterator)
      return tf_agent.train(experience)

    if self.use_tf_functions:
      train_step = common.function(train_step)

    global_step_val = global_step.numpy()

    while global_step_val < self.num_iterations:

      start_time = time.time()
```

```
      time_step, policy_state = collect_driver.run(time_step=time_step,
                                                   policy_state=policy_state,)

      for _ in range(self.train_steps_per_iteration):
        train_loss = train_step()
        time_acc += time.time() - start_time

      global_step_val = global_step.numpy()

      if global_step_val % self.log_interval == 0:

        logging.info('step = %d, loss = %f', global_step_val, train_loss.loss)

        steps_per_sec = (global_step_val - timed_at_step) / time_acc

        logging.info('%.3f steps/sec', steps_per_sec)

        tf.compat.v2.summary.scalar(name='global_steps_per_sec',
                                    data=steps_per_sec,
                                    step=global_step)

        timed_at_step = global_step_val

        time_acc = 0

      for train_metric in train_metrics:
        train_metric.tf_summaries(train_step=global_step,
                                  step_metrics=train_metrics[:2])

      if global_step_val % self.eval_interval == 0:
        results = metric_utils.eager_compute(
                        eval_metrics,
                        eval_tf_env,
                        eval_policy,
                        num_episodes=self.num_eval_episodes,
                        train_step=global_step,
                        summary_writer=eval_summary_writer,
                        summary_prefix='Metrics',)

      if global_step_val % self.log_interval == 0:
        metric_utils.log_metrics(eval_metrics)

      if global_step_val % self.train_checkpoint_interval == 0:
        train_checkpointer.save(global_step=global_step_val)

      if global_step_val % self.policy_checkpoint_interval == 0:
        policy_checkpointer.save(global_step=global_step_val)

      if global_step_val % self.replay_buffer_checkpoint_interval == 0:
         replay_buffer_checkpointer.save(global_step=global_step_val)

    video_file_name = os.path.join(self.root_dir, self.name + ".mp4")
    self._create_video(video_filename = video_file_name,
                       py_env = py_env,
                       agent=tf_agent)

    return train_loss

  def _create_video(self,
                    video_filename="video.mp4",
                    num_episodes = 3,
                    py_env=None,
```

```
                    agent=None):
        """Create a video for a few episodes
           Attribution:
             This method has been adapted from the example code provided by
               Holt, S. (2020). Value and Policy based agents:
https://pathtopioneer.com/blog/2020/07/rl-4
               licensed under a Creative Commons CC BY-NC-SA 4.0 license.
        """
        env = tf_py_environment.TFPyEnvironment(py_env)
        with imageio.get_writer(video_filename, fps=15) as video:
            for _ in range(num_episodes):
                time_step = env.reset()
                video.append_data(py_env.render())
                while not time_step.is_last():
                    action_step = agent.policy.action(time_step)
                    time_step = env.step(action_step.action)
                    video.append_data(py_env.render())

    def _get_the_agent(self,
                       time_step_spec,
                       observation_spec,
                       action_spec, global_step,
                       debug_summaries,
                       summarize_grads_and_vars) -> TFAgent:

        # Subclasess must override this method"
        raise ValueError("This is a base class, you must implement a subclass
and override this method")
```

12.2 Codi font del submòdul *utils.py* del mòdul *agents.naf*

```
import tensorflow as tf
from tf_agents.networks import network
from tf_agents.networks import utils as networks_utils
from tf_agents.networks import actor_distribution_network

class CompositeNetwork(network.Network):
    def __init__(self,
        input_tensor_spec,
        output_layer_params=1,
        fc_layer_params=(75, 40),
        dropout_layer_params=None,
        activation_fn=tf.keras.activations.relu,
        kernel_initializer=None,
        batch_squash=True,
        dtype=tf.float32,
        name='CompositeNetwork'):

        super(CompositeNetwork, self).__init__(
            input_tensor_spec=input_tensor_spec,
            state_spec=(),
            name=name)

        self._shared_network = None

        self._hidden_layers = networks_utils.mlp_layers(
            fc_layer_params=fc_layer_params,
            dropout_layer_params=dropout_layer_params,
            activation_fn=activation_fn,
            kernel_initializer=kernel_initializer,
            )

        self._output_layer = tf.keras.layers.Dense(
            output_layer_params,
            activation=None,
            kernel_initializer=tf.compat.v1.initializers.random_uniform(
                minval=-0.03, maxval=0.03))

    def set_shared_network(self, shared_network):
        if not self.built:
            if shared_network.built:
                self._shared_network = shared_network
            else:
                raise ValueError("The shared network must have been built")
        else:
            raise ValueError("You cannot initialize the network after being
built")

    def call(self, observation, step_type=None, network_state=(),
training=False):
        logits = observation
        current_network_state = network_state

        if self._shared_network is not None:
            logits, current_network_state = self._shared_network(logits,
step_type=step_type, network_state=current_network_state, training=training)

        for layer in self._hidden_layers:
            logits = layer(logits, training=training)

        logits = self._output_layer(logits, training=training)
```

```
        return logits, current_network_state

    def create_variables(self, input_tensor_spec=None, **kwargs):
        next_layer_input_spec = None
        if self._shared_network is None:
            next_layer_input_spec = input_tensor_spec
        super(CompositeNetwork,
self).create_variables(input_tensor_spec=next_layer_input_spec, **kwargs)

class CompositeValueNetwork(CompositeNetwork):
    def __init__(self,
        input_tensor_spec,
        fc_layer_params=(75, 40),
        dropout_layer_params=None,
        activation_fn=tf.keras.activations.relu,
        kernel_initializer=None,
        batch_squash=True,
        dtype=tf.float32,
        name='CompositeValueNetwork'
        ):
        super(CompositeValueNetwork, self).__init__(
            input_tensor_spec,
            output_layer_params=1,
            fc_layer_params=fc_layer_params,
            dropout_layer_params=dropout_layer_params,
            activation_fn=activation_fn,
            kernel_initializer=kernel_initializer,
            batch_squash=batch_squash,
            dtype=dtype,
            name=name
            )

    def call(self, observation, step_type=None, network_state=(),
training=False):
        logits, current_network_state = super(CompositeValueNetwork,
self).call(observation, step_type=step_type, network_state=network_state,
training=training)
        logits = tf.squeeze(logits, -1)
        return logits, current_network_state

class CompositeLNetwork(CompositeNetwork):
    def __init__(self,
        input_tensor_spec,
        action_spec,
        fc_layer_params=(75, 40),
        dropout_layer_params=None,
        activation_fn=tf.keras.activations.relu,
        kernel_initializer=None,
        batch_squash=True,
        dtype=tf.float32,
        name='CompositeValueNetwork'
        ):

        # action_spec_size = Number of terms of the action space
        action_spec_size = tf.reduce_prod(action_spec.shape).numpy()

        # the number of terms o the L-Matrix is defined by the sum of the
succession
        # from 1 to the number of terms of the action space, this is:
        l_network_num_terms = int(action_spec_size * (action_spec_size + 1) /
2)
```

```
        super(CompositeLNetwork, self).__init__(
            input_tensor_spec,
            output_layer_params=l_network_num_terms,
            fc_layer_params=fc_layer_params,
            dropout_layer_params=dropout_layer_params,
            activation_fn=activation_fn,
            kernel_initializer=kernel_initializer,
            batch_squash=batch_squash,
            dtype=dtype,
            name=name
            )

class CompositeActorDistributionNetwork(network.Network):
    def __init__(self,
        input_tensor_spec,
        output_tensor_spec,
        fc_layer_params=(200, 100),
        dropout_layer_params=None,
        activation_fn=tf.keras.activations.relu,
        kernel_initializer=None,
        batch_squash=True,
        dtype=tf.float32,

discrete_projection_net=actor_distribution_network._categorical_projection_net,

continuous_projection_net=actor_distribution_network._normal_projection_net,
        name='CompositeActorDistributionNetwork'):

        super(CompositeActorDistributionNetwork, self).__init__(
            input_tensor_spec=input_tensor_spec,
            state_spec=(),
            name=name)

        self._shared_network = None


        self._ad_network_params = {
            "output_tensor_spec": output_tensor_spec,
            "fc_layer_params": fc_layer_params,
            "dropout_layer_params": dropout_layer_params,
            "activation_fn": activation_fn,
            "kernel_initializer": kernel_initializer,
            "batch_squash": batch_squash,
            "dtype": dtype,
            "discrete_projection_net": discrete_projection_net,
            "continuous_projection_net": continuous_projection_net
            }


        self._ad_network = None

    def initialize(self, shared_network):
        if not self.built:
            if shared_network.built:
                self._shared_network = shared_network
            else:
                raise ValueError("The shared network must have been built")
        else:
            raise ValueError("You cannot initialize the network after being
built")
```

```
    def call(self, observation, step_type=None, network_state=(),
training=False):
        logits = observation
        current_network_state = network_state

        if self._shared_network is not None:
            logits, current_network_state = self._shared_network(logits,
step_type=step_type, network_state=current_network_state, training=training)

        logits, current_network_state = self._ad_network(logits,
step_type=step_type, network_state=current_network_state, training=training)

        return logits, current_network_state

    def create_variables(self, input_tensor_spec=None, **kwargs):

        self_input_spec = None

        if self._shared_network is None:

            self_input_spec = input_tensor_spec

            if input_tensor_spec is None:
                ad_network_input_spec = self.input_tensor_spec
            else:
                ad_network_input_spec = input_tensor_spec

        else:
            ad_network_input_spec = self._shared_network._network_output_spec

        self._ad_network = actor_distribution_network.ActorDistributionNetwork(
            ad_network_input_spec,
            self._ad_network_params["output_tensor_spec"],
            fc_layer_params=self._ad_network_params["fc_layer_params"],

dropout_layer_params=self._ad_network_params["dropout_layer_params"],
            activation_fn=self._ad_network_params["activation_fn"],
            kernel_initializer=self._ad_network_params["kernel_initializer"],
            batch_squash=self._ad_network_params["batch_squash"],
            dtype=self._ad_network_params["dtype"],

discrete_projection_net=self._ad_network_params["discrete_projection_net"],

continuous_projection_net=self._ad_network_params["continuous_projection_net"],
            name="ActorDistributionNetwork"
            )


self._ad_network.create_variables(input_tensor_spec=ad_network_input_spec,
**kwargs)

        super(CompositeActorDistributionNetwork,
self).create_variables(input_tensor_spec=self_input_spec, **kwargs)
```

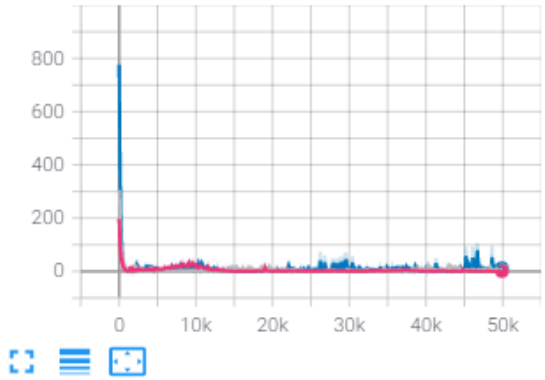
12.3 Dependències del producte desenvolupat

El producte desenvolupat funciona sobre Python 3.7 i depèn de les següents biblioteques:

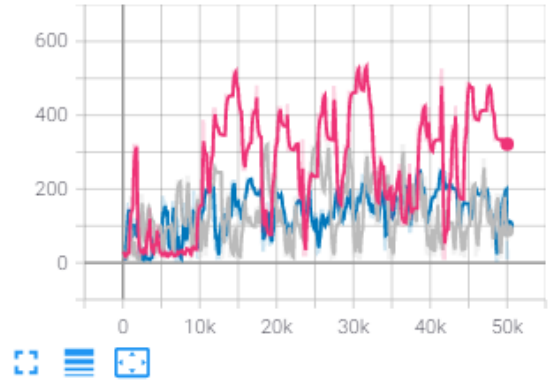
- tf_agents (0.6.0)
- tensorflow (2.3.1)
- tensorflow-probability (0.11.1)
- tensorboard (2.4.0)
- gym (0.17.3)
- numpy (1.18.5)
- imageio (2.5.0)

12.4 Exemple d'output de TensorBoard per a les traces de l'agent NAF

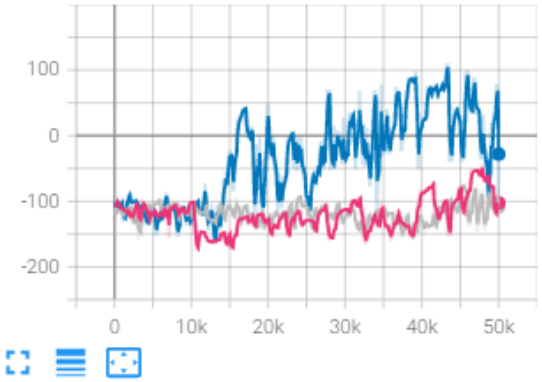
Losses/avg_loss
tag: Losses/avg_loss



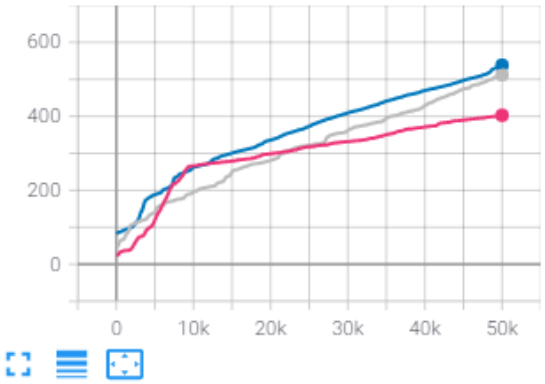
Metrics/AverageEpisodeLength
tag: Metrics/AverageEpisodeLength



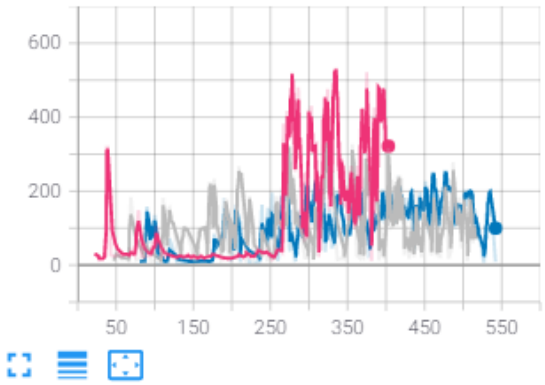
Metrics/AverageReturn
tag: Metrics/AverageReturn



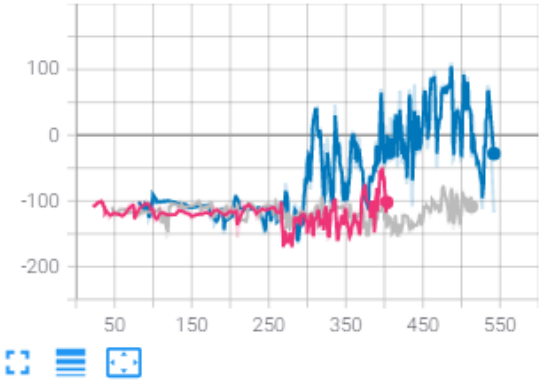
Metrics/NumberOfEpisodes
tag: Metrics/NumberOfEpisodes



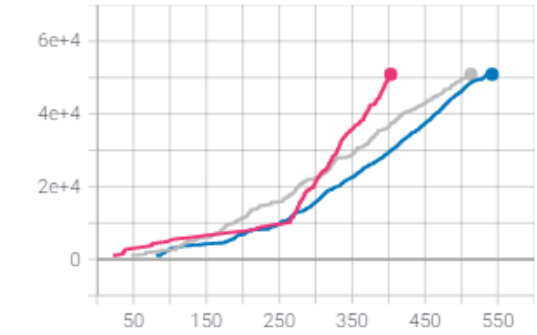
Metrics_vs_NumberOfEpisodes/AverageEpisodeLength
tag: Metrics_vs_NumberOfEpisodes/AverageEpisodeLength



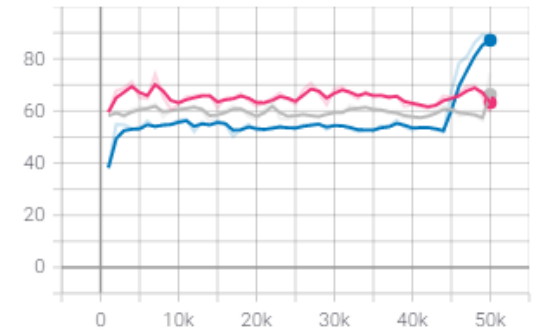
Metrics_vs_NumberOfEpisodes/AverageReturn
tag: Metrics_vs_NumberOfEpisodes/AverageReturn



Metrics_vs_NumberOfEpisodes/EnvironmentSteps
tag: Metrics_vs_NumberOfEpisodes/EnvironmentSteps



global_steps_per_sec
tag: global_steps_per_sec



12.5 Relació d'experiments proporcionats i paràmetres emprats

Els paràmetres per defecte per als experiments han estat els següents:

```
# OFF_POLICY TIME_STEP BASED EXPERIMENT GENERIC HYPERPARAMETERS
num_eval_episodes = 3
summary_interval = 100
env_name = "BipedalWalker-v3"
env_action_repeat_times = 4
debug_summaries=False
summarize_grads_and_vars=False
replay_buffer_capacity=5000
initial_collect_steps=1000
collect_steps_per_iteration=1
use_tf_functions=True
sample_batch_size = 256
num_iterations=50000
train_steps_per_iteration=1
log_interval=1000
eval_interval=1000
train_checkpoint_interval=5000
policy_checkpoint_interval=5000
replay_buffer_checkpoint_interval=5000

# SAC AND NAF ANALOG HYPERPARAMETERS
import tensorflow as tf
learning_rate=3e-4
target_update_tau=0.005
target_update_period=1
td_errors_loss_fn=tf.math.squared_difference
gamma=0.99


# NAF SPECIFIC HYPERPARAMETERS
from tf_agents.networks import encoding_network
naf_root_dir = os.path.join(root_dir, "naf")
preprocessing_conv_layer_params = None
preprocessing_conv_type = encoding_network.CONV_TYPE_1D
preprocessing_fc_layer_params = (256, )
preprocessing_dropout_layer_params = None
v_network_fc_layer_params = (256, )
v_network_dropout_layer_params = None
l_network_fc_layer_params = (256, )
l_network_dropout_layer_params = None
policy_network_fc_layer_params = (256, )
policy_network_dropout_layer_params = None
policy_network_uses_shared_preprocessing_network = True
noise_factor=0.1

# SAC SPECIFIC HYPERPARAMETERS

sac_root_dir = os.path.join(root_dir, "sac")
actor_fc_layers=(256, 256)
critic_obs_fc_layers=None
critic_action_fc_layers=None
critic_joint_fc_layers=(256, 256)
actor_learning_rate=learning_rate
critic_learning_rate=learning_rate
alpha_learning_rate=learning_rate
reward_scale_factor=0.1
gradient_clipping=None
```

A continuació es proporciona la relació d'experiments per als quals s'ha annexat traces i vídeos. En cada experiment s'indica els paràmetres que s'han modificat.

Sèrie sac_times_#

Paràmetres comuns a tota la sèrie:

```
Classe de l'experiment: SacExperiment
```

Paràmetres específics de cada experiment:

Experiment	Paràmetres modificats
# = 1	`env_action_repeat_times = 1`
# = 2	`env_action_repeat_times = 2`
# = 3	`env_action_repeat_times = 3`
# = 4	`env_action_repeat_times = 4`
# = 5	`env_action_repeat_times = 5`

Sèrie sac_env_hardcore_times_#

Paràmetres comuns a tota la sèrie:

```
Classe de l'experiment: SacExperiment

env_name = "BipedalWalkerHardcore-v3"
```

Paràmetres específics de cada experiment:

Experiment	Paràmetres modificats
# = 1	`env_action_repeat_times = 1`
# = 2	`env_action_repeat_times = 2`
# = 3	`env_action_repeat_times = 3`
# = 4	`env_action_repeat_times = 4`
# = 5	`env_action_repeat_times = 5`
# = 6	`env_action_repeat_times = 6`
# = 7	`env_action_repeat_times = 7`
# = 8	`env_action_repeat_times = 8`

Sèrie sac_env_hardcore_times_#_nets_256_128_64_rbc_50k_iter_100k

Paràmetres comuns a tota la sèrie:

```
Classe de l'experiment: SacExperiment

env_name = "BipedalWalkerHardcore-v3"
num_iterations = 100000
replay_buffer_capacity = 50000
actor_fc_layers = (256, 128, 64)
critic_joint_fc_layers = (256, 128, 64)
```

Paràmetres específics de cada experiment:

Experiment	Paràmetres modificats
# = 1	`env_action_repeat_times = 1`
# = 4	`env_action_repeat_times = 4`
# = 8	`env_action_repeat_times = 8`

Sèrie naf_times_#

Paràmetres comuns a tota la sèrie:

```
Classe de l'experiment: NafExperiment
```

Paràmetres específics de cada experiment:

Experiment	Paràmetres modificats
# = 4	`env_action_repeat_times = 1`
# = 6	`env_action_repeat_times = 2`
# = 8	`env_action_repeat_times = 3`

Sèrie naf_times_8_replica#

Paràmetres comuns a tota la sèrie:

```
Classe de l'experiment: NafExperiment

env_action_repeat_times = 8
```

Paràmetres específics de cada experiment:

Experiment	Paràmetres modificats
# = 1	-
# = 2	-
# = 3	-

Sèrie naf_times_8_noise_x#

Paràmetres comuns a tota la sèrie:

```
Classe de l'experiment: NafExperiment

env_action_repeat_times = 8
```

Paràmetres específics de cada experiment:

Experiment	Paràmetres modificats
# = 2	`noise_factor=0.2`
# = 4	`noise_factor=0.4`

Sèrie naf_times_8_replay_buffer_x#

Paràmetres comuns a tota la sèrie:

```
Classe de l'experiment: NafExperiment

env_action_repeat_times = 8
```

Paràmetres específics de cada experiment:

Experiment	Paràmetres modificats
# = 2	`replay_buffer_capacity = 10000`
# = 4	`replay_buffer_capacity = 20000`

Sèrie naf_times_8_tf_#_mode

Paràmetres comuns a tota la sèrie:

```
Classe de l'experiment: NafExperiment

env_action_repeat_times = 8
```

Paràmetres específics de cada experiment:

Experiment	Paràmetres modificats
# = eager	`use_tf_functions=False`
# = graph	`use_tf_functions=True`

Sèrie naf_env_hardcore_times_#

Paràmetres comuns a tota la sèrie:

```
Classe de l'experiment: NafExperiment

env_name = "BipedalWalkerHardcore-v3"
```

Paràmetres específics de cada experiment:

Experiment	Paràmetres modificats
# = 4	`env_action_repeat_times = 4`
# = 8	`env_action_repeat_times = 8`

Experiment rnd_times_8

```
Classe de l'experiment: RndExperiment

env_action_repeat_times = 8
```