eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Informatika Ingeniaritza Gradua

Konputazioa

Gradu Amaierako Lana

Ikasketa automatiko teknikak Super Mario bideo-jokoan

Egilea

Iker Zabala

2020

eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Informatika Ingeniaritza Gradua

Konputazioa

Gradu Amaierako Lana

Ikasketa automatiko teknikak Super Mario bideo-jokoan

Egilea
Iker Zabala

Zuzendaria
Jesús Ibáñez

2020

Laburpena

Proiektu honen helburu nagusia **Super Mario (NES)** joko klasikoan jokatzen ikasten duen hainbat ikasketa automatiko algoritmo inplementatzea da, urteetan zehar izandako hobekuntzak eta desberdintasunak ikusiz. Horretarako errefortzu bidezko ikasketa automatiko teknikak aplikatuko dira, oso egokiak direnak bideo-jokoekin erabiltzeko. Lehenik q-learning algoritmoa inplementatuko da, 1989 urtean sortu zena, eta ondoren algoritmo honen eboluzioa erabiliko da, deep q-network izenez ezagutzen dena. Azken algoritmo honek q-learning sare neuronal batekin konbinatzen du.

Bi algoritmo hauen artean 30 urteko diferentzia dago, beraz emaitzen hobekuntza ikustea espero da, denbora asko izan duelako eboluzionatzeko eta arazoei irtenbidea aurkitzeko.

Proiektu hau hainbat fasetan banatu da:

- Planifikazioa egin eta erabiliko diren tresnak eta metodologia aukeratu.
- Q-learning inplementatu eta hobetu.
- Deep q-learning inplementatu, aurreko algoritmoaren eta sare neuronalen konbinazioa dena.
- Emaitzak konparatuz ondorioak atera.

Aurkibidea

List Of Images

Sarrera

Adimen artifiziala bideo-jokoen parte izan dira hauek sortu zirenetik, eta 1978 urtean garrantzi gehiago hartzen hasi zen arkade makina famatuak azaldu zirenean (Space invaders, asteroids…). Pacman jokoan adibidez, etsai bakoitzak portaera propioa dauka.

Bideo-jokoetan erabiltzen diren teknikak eta teknika akademikoak desberdintasun bat dute: Jokoetan erabiltzailearen esperientzia hobetzeko erabiltzen dira, eta ez dago ikasketa automatikorik. NPC batek pathfinding algoritmoa erabiltzen du mapan mugitzeko adibidez, edo erabaki automata finitu sinpleak dituzte, adimena simulatuz. Honen arrazoia prozesaketa arindu nahi delako da, algoritmo sinpleak sortzen dira jokoetan eta alde batera utzi ikasketa automatikoa. Hasieran teknika sinpleak erabiltzen ziren, gehien bat hardware mugengatik, baina urteetan zehar teknikak hobetzen joan dira. Proiektu honetan bideo-jokoek erabiltzen duten adimen artifiziala alde batera utziko da, ikasketa automatikoan gehiago enfokatuz.

Bideo-jokoak eta adimen artifizialak aldi berean eboluzionatu dute teknologiarekin. Ordenagailuak geroz eta indartsuagoak izanda gauza konplexuagoak sortzea ahalbidetu du. Sare neuronalen ideia 1943 jaio zen eta 1958 urtean perzeptroi ordenagailua sortu zen, lehen sare neuronalaren inplementazioa izan zena.

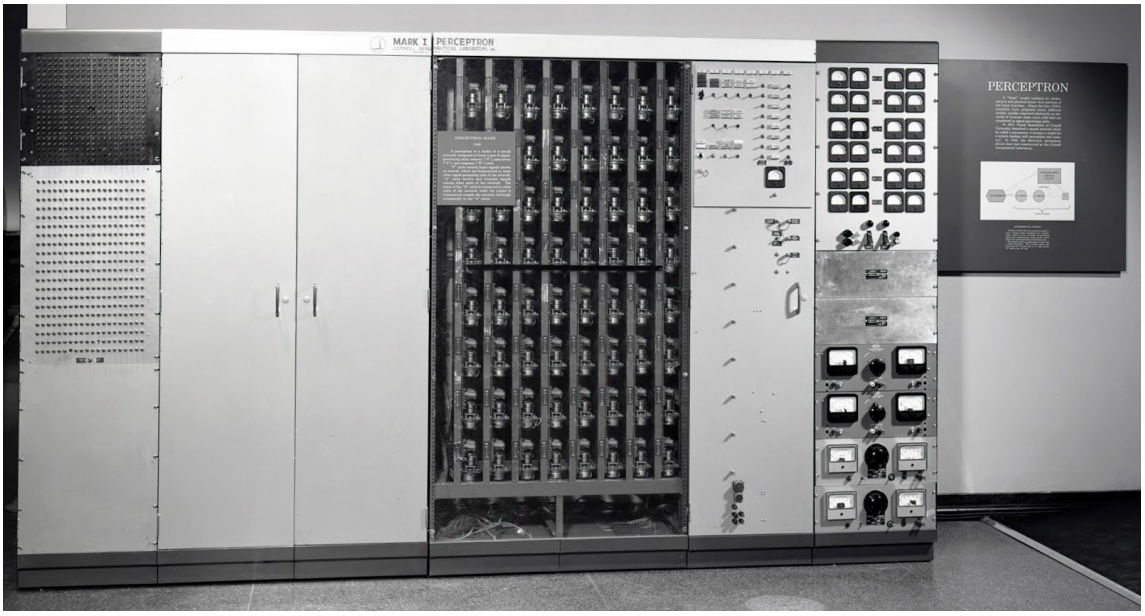


Figure 1. Mark I pertzeptroi makina

Errefortzu bidezko ikasketa berriz gutxienez 1930 urtean existitzen zen (Monte Carlo metodoa) baina oraindik ez zegoen oso garatuta. 1950 urtean Alan Turingek algoritmo genetikoa proposatu zuen, bakarrik ikasten zuen makina bat sortzeko. Manhattan proiektuan Monte Carlo algoritmoa erabili zen hidrogeno bonbaren garapenean. Fisika eta kimika simulazioetan erabili zen ere, eta ikerkuntza operatibo ere sortzea ahalbidetu zuen Munduko Bigarren Gerran, erabaki taktiko eta estrategikoak hartzeko. Garai hartan izan zuen aplikazioa militarra izan zen, oraindik ordenagailu pertsonalak existitzen ez zirelako eta oso makina garestiak eta handiak zirelako, baina gaur egun beste arloetan ere erabiltzen da, industrian eta ekonomian adibidez.

Urteetan zehar, mahai jokoetan arrakasta lortu duten programak sortu dira. 1992 urtean TD-Gammon izeneko sare neuronala sortu zuen IBM-k, jokalari profesionalen antzeko maila lortzen zuenak.

1997 urtean Deep Blue makinak xakean Kasparov, munduko txapelduna, menperatzea lortu zuen. Hau egiteko alfa-beta algoritmoa hainbat optimizazio teknikekin erabiltzen zuen. Konputazio behar handiak zituen eta ez zuen ezer berririk ikasten esperientziatik, honen ondorioz egile batek hau ez zela adimen artifiziala esaten zuen.

18 urte beranduago, 2015ean, AlphaGo programak Go munduko txapelduna garaitzea lortu zuen, xakea baina konplexuagoa den joko bat. Honek Monte Carlo zuhaitz bilaketa erabiltzen zuen, eta bilaketa hori geroz eta hobea izateko sare neuronal batekin konbinatu zen.

Mahai jokoetan emaitza onak lortu eta gero, hurrengo helburua denbora errealean exekutatzen diren inguruneekin ikastea izango litzateke. Ingurune hauek orokorrean konplexuagoak dira, eta bizitza errealean aplikagarriak diren hainbat eremuetan erabilgarriak izan daitezke, adibidez atea irekitzen duen robotak edo kotxeen gidatze automatikoa.

Bideo-jokoak oso ingurune egokiak dira aurrerapen hauek egiteko, orokorrean denbora errealean gertatzen direlako eta informazioa atzigarria delako mundu errealeko sentsorerik behar izan gabe. Grand Theft Auto jokoa nahiko fotoerrealista denez kotxeak gidatzen ikasten duen algoritmo bat sortzeko erabili daiteke. Mota guztietako inguruneak aurkitu ditzakegu, batzuk oso sinpleak direnak (hiruko artzain-jokoa), eta baita ere oso konplexuak direnak (Kerbal Space Program, fisika simulatzen duena espazio-ontziak orbitan jartzeko edo beste planetak esploratzeko). Minecraft jokoan adibidez, posiblea da zirkuituak eraikitzea blokeak erabiliz, eta hau Turing konputagarria dela baieztatu da. Honek ingurune simulatu batean ikasketa automatiko teknikak probatzeko infinitu aukera ematen ditu.

Sare neuronalen aplikazioen zerrenda[1] oso handia da, eta bi ikuspuntu daude teknologia honi buruz. Batzuek edozein arazo mota ebazteko gai den tresna bat dela uste dute, ikasketa automatikoa betirako aldatuko duena. Beste batzuek joera iragankor bat dela eta ez dela hainbesterako.

[1] https://medium.com/towards-artificial-intelligence/main-types-of-neural-networks-and-its-applications-tutorial-734480d7ec8e

Aurrekariak

Atal honetan zein faktorek izan duten eragin proiektuaren hasieraketan azalduko dira. Proiektuaren eremuan zein faktorek sortu zuten interesa azalduko da, baita ere aurrekari teknologikoak.

Aurrekari teknologikoak

Sare neuronalak erabiliz hamarkada honen hasieran aplikazio berriak sortzeko balio duela ikusi da, adibidez konputazio bidezko ikusmena edo ahozko hizkuntzaren antzematea. Gainbegiratutako eta gainbegiratu gabeko ikasketan arrakasta izan zuen, baina ez zegoen oso garbi errefortzu bidezko ikasketan zein emaitza lortuko ziren, erronka bat suposatzen zuen.

2013 urtean DeepMind enpresak Atari bideo-jokoetan ikasten zuen sare neuronala erabili zuen, q-learning algoritmoarekin nahasiz. Guztira zazpi joko erabili zituzten, orokorrean nahiko sinpleak zirenak, eta sarrera moduan pixel balioak erabiltzen ziren. Horietatik bost jokoetan algoritmoak pertsona batek baino puntuazio hobea lortzen zuen, errefortzu bidezko ikasketa eta sare neuronala batera erabiltzeko aukera dagoela erakutsiz.

2017 urtean aurrerakuntzak independenteak egin ziren, eta DeepMind-ek hauetako sei hobekuntza konbinatuz algoritmo originala hobetzea lortu zuten. Algoritmo berri hau “Rainbow” izenez ezagutzen da.

2019 urtean OpenAI enpresak Dota 2 jokoan jolasten ikasten zuen programa sortu zuen. Joko hau oso konplexua da hainbat arrazoiengatik:

- Partidek asko irauten dute, lau frametik bat erabiliz 20.000 pauso inguru irauten du, edo 45 minutu (xakea partidek 80 mugimendu dituzte batezbestekoan, eta GO partidek 150).

- Ingurunearen zati bat bakarrik ikusi daiteke, maparen zati handia estalita egonda. Informazio falta dagoenean erabakiak hartu behar dira, etsaiaren portaera zein izango den asmatzen saiatuz

- Egoera eta ekintza espazioaren dimentsiok handiak dira. Pauso bakoitzean algoritmoaren egoera 16.000 zenbaki errealez osatuta dago (xakean 1000 balio dira, eta Go jokoan 6.000 zenbaki bitar). Ekintza espazioa berriz, 8.000-18.000 tartean dago (xakean 35 ekintza posible, eta Go jokoan 250 inguru).

Aurreko arazoei aurre egiteko ezinbestekoa izan zen erabilitako hardwarea eskala handi batera eramatea. Programak ondo jokatzen ikasteko milaka GPU erabili ziren eta hainbat hilabete egon ziren entrenatzen. Informazio bisuala erabili beharrean (pixel balioak) array

moduan jasotzen zuen behar zuen informazio guztia. Jokoaren exekuzioa azeleratuz eta hamar hilabete entrenatzen egon ondoren (guztira milioika urte izan direnak ordenagailuarentzat) munduko txapeldunak menperatzea lortu zuen, ataza zailetan entrenamendua posiblea dela erakutsiz. Proiektu horretan erabiltzen diren teknikak ez dira proiektu honetan aplikatuko printzipioz, ez dutelako pixel balioak erabiltzen input moduan eta hardware diferentzia handia dagoelako ere.

Eremuan interesa

Bideo-jokoen eremua gazte nintzenetik iruditu zait oso interesgarria. Oraindik gogoratzen dut Crash Bandicoot jokoaren atzeko logikak nola funtzionatzen zuen asmatzen saiatzen nintzela. Nire teoria frame posible guztiak irudi moduan gordeta zeudela zen, eta automata finitu erraldoi bat erabiliz ekintza baten ondoren datorren irudia zein izango zen kalkulatzen zela.

Jokoei esker hasi nintzen informatika munduan. 15-16 urte inguru izanda Lua lengoaia ikasten hasi nintzen eta PSP kontsolan exekutatu nuen nire lehen HelloWorld programa eta oso sinplea zen joko bat. Proiektu honetan Lua erabiltzen denez honek nostalgia puntu bat izan du.

Adimen artifizialak beti izan du interes apur bat ere, hauek bideo-jokoen parte izan dira hauek sortu zirenetik. Interes hau asko handitu zen honek zuen potentziala ikusi nuenean hainbat irakasgaietan. Zoragarria iruditu zitzaidan Shannon-en entropia ikastea, eta nola teoria hori Akinator moduko aplikazio bat sortzeko gai zen. Aurrerago Weka erabili ahal izan nuen ere gainbegiratutako ikasketa aplikatzeko, oso interesgarria izan zen irudi sailkatzaile bat eraikitzeko dauden metodoak ikastea.

Proiektu honetan erabiltzen diren algoritmoak (errefortzu bidezko ikasketa eta sare neuronalak) berriak izan dira niretzat, eta proiektu hau amaitzean puzzlean falta zen pieza bat jartzea bezala dela iruditzen zait.

1. Plangintza

1.1 Irismena

1.1.1 Proiektuaren helburu zehatzen deskribapena

Proiektu honen helburu nagusia errefortzu bidezko algoritmoaren eboluzioa aztertzea izango da. Horretarako lehendabizi q-learning inplementatuko da, 1989 urtean sortu zena. Ondoren q-learning algoritmoaren bertsio modernoago bat erabiliko da, deep q-network izenekoa, lehen algoritmoa eta sare neuronal bat konbinatzen dituenak. 25 urte inguruko jauzia dago bi teknika hauen artean, horregatik bigarren bertsioa hobea izatea espero da. Bi hauen arteko desberdintasun nagusiak aztertuko dira, eta bakoitzaren mugak.

Probak egiteko Super Mario (NES) joko klasikoa erabiliko da ingurune moduan, Atari jokoak baino konplexuagoa da eta hardware eskakizunak exekuzio garaian ez dira handiak izango, garrantzizkoa izango dena entrenamenduaren abiadura egokia izateko.

Algoritmoak jasotzen duen informazio pertsona batek antzematen duenaren antzekoa izango da. Horregatik pixel balioak prozesatuko dira ekintza hoberena zein izango den jakiteko, RAM balioak baztertuz.

Algoritmoaren kalitatea neurtzeko hainbat irizpide erabiliko dira:

- Programak sortzen duen lan zama. Jokoa denbora errealean exekutatuko denez, algoritmoa arina izatea baloratuko da, exekuzioa azkarragoa izateko eta ikasketa ez geldotzeko. Emuladorearen software azelerazioa erabiltzeko asmoa dago, jokoa abiadura azkarrean exekutatzeko eta azkarrago ikasteko.

- Programaren trebetasuna mailak gainditzeko. Algoritmoaren eraginkortasuna hainbat mailetan probatuko da, maila asko pasatzen baditu hobea izango da. Jokoak 32 maila ditu guztira.

- Maila pasatzeko behar duen denbora, edo aurkitutako soluzioaren kalitatea.

- Egoera berri edo ezezagunei aurre egiteko ahalmena ere baloratuko da.

1.1.2 Eskakizunak

- Errefortzu bidezko algoritmoaren inplementazioa. Q-learning algoritmoa inplementatuko da Lua erabiliz, Markov Erabaki Prozesua eta Bellman ekuazioa erabiltzen duena.

- Sare neuronala. Q-learning inplementatu ondoren, sare neuronal batekin konbinatuko da zein izango den egin beharreko ekintza aukeratzeko.

- Plangintza. Modu zehatz batean definituko da plangintza. Bertan irismena, arriskuak, denbora kudeaketa etab jasoko dira.

- Dokumentazioa. Proiektuari buruzko informazio guztia bertan azalduko da.

- Defentsa egiteko aurkezpen dokumentua.

1.1.3 Baztertutako helburuen deskribapena

Puntuazio hoberena lortzeko helburua baztertua izango da. Puntuazio hau handitzeko Mariok hainbat ekintza egin ditzake, adibidez txanponak hartu edo etsaiak garaitu.

Maila pasatzeko modu azkarrena aurkitzearen helburua ere baztertua izango da. Hau lortzea oso konplexua denez emaitza azpi-optimoak onartuko dira maila pasatzeko.

1.1.4 Tresna eta teknologien deskribapena

Atal honetan proiektuan zehar erabili diren tresna eta teknologia guztiak azaltzen dira.

Jokoa exekutatu eta kontrolatzeko **Bizhawk** [2] tresna erabili da, hainbat kontsola emulatzeko kapaza dena. TAS (Tool Assisted Speedrun) komunitate barruan ezaguna da, ekintza sekuentziak exekutatzeko gai delako. Hainbat funtzio erabilgarri ditu, frame bakoitza konputatzeko aukera du, RAM memoriako datuak irakurri, jokoaren exekuzioa azeleratu, savestate-ak erabili (jokoaren egoera edozein unetan gordetzeko eta kargatzeko aukera), pantailaren argazki bat lortu eta hainbat gauza gehiago. Lua script-ak exekutatzen ditu ere, honi esker programa bat egin daiteke emuladorearen egoera kontrolatzeko. Bizhawkek desabantaila bat du, Windows sistema eragilean ibiltzeko sortua delako. Posiblea da Linux erabiltzea ere, baina aplikazioa ez da egonkorra eta errore edo arazo gehiago azaldu daitezke.

Beraz, jokoan ibiltzeko eta honen informazioa atzitzeko **Bizhawk** eta **Lua** lengoaia erabiliko dira, **Windows** sistema eragilearekin. Portatil pertsonala erabili dut probak egiteko, Asus TUF FX705GM modeloa. Honen prozesadorea Intel Core i7-8750H bat da, 16GB RAM ditu eta txartel grafikoa NVIDIA GeForce GTX 1060, 6GB RAM izanik.

[2] http://tasvideos.org/BizHawk.html

Lua lengoaiak muga batzuk dituenez, **Python** programazio lengoaia ere erabiliko da. Python lengoaiaren abantailen artean, liburutegiak erabiltzeko erraztasuna izan da faktore handiena:

- **Pytorch**[3]: Sare neuronala sortzeko erabiliko da.
- **PIL**[4] liburutegia: Irudiak kargatzeko eta prozesatzeko hainbat aukera ditu: gris eskalara transformatu, irudiak hainbat modutan atzitu (clipboard bidez, png fitxategi bat, byte kate bat…), irudiaren tamaina aldatu.
- **Hashlib** liburutegia: SHA256 erabiliz memoria aurrezteko erabiliko da kriptografia liburutegi hau.
- Hainbat matematika liburutegi (**numpy**, **random**, **deque** lista egitura…).
- **Http.server**, **socket** eta **json** liburutegiak: Lua eta Python arteko komunikaziorako erabiliko dira.
- **Matplotlib** liburutegia: Emaitzen grafikak pantailaratzen laguntzen du.

Pytorch-ek txartel grafikoa erabili dezan **CUDA** [5] instalatu behar da ere, bestela CPU bidez egingo dira kalkuluak, mantsoagoa dena.

1.1.5 Erabilitako metodologia

Zein algoritmo mota erabili behar den erabakitzeko garrantzitsua da aukerak zeintzuk diren jakitea. Hiru adar nagusietan banatzen dira ikasketa automatiko teknikak:

- **Gainbegiraturiko ikasketa**. Etiketatuta dagoen informazioa erabiltzen du entrenatzeko, ondoren kasu berriak sailkatzeko (argazki batean katu bat edo txakur bat agertzen den, adibidez).
- **Gainbegiratu gabeko ikasketa**. Aurrekoaren antzekoa da, baina sarrerako informazioa etiketatu gabe dago. Informazio hori erabiliz egiturak edo ezkutuan dauden eredu konplexuak aurkitzeko gai da, kluster analisiak egiteko adibidez.
- **Errefortzu bidezko ikasketa**. Bertan, egile batek ingurune dinamiko batean metatutako sari handiena lortzeko zein ekintza egin behar dituen ikasten du, lortutako esperientziatik. Sari hau orokorrean atzerapen batekin dator, ez da ekintza egiten den unean lortzen, baizik eta ekintza kate konkretu baten ondoren. Honen helburu nagusia ikasitako informazioaren erabilpena eta oraindik miatu gabeko egoeren oreka aurkitzea da, atzerapenekin datozen sari horiek lortzeko. Orokorrean erraza da ekintzak ebaluatzea, baina zaila zehaztea. Ibiltzen ikasi behar duen robot batek adibidez, honen posizioa edo abiadura erabili daiteke emaitzak onak ala txarrak diren jakiteko, baina zaila da zehaztea zein ekintza egin

[3] https://pytorch.org/
[4] https://pillow.readthedocs.io/
[5] https://developer.nvidia.com/cuda-10.1-download-archive-base?

behar dituen. Biologia imitatuz gauzak desberdinak probatuko ditu eta azkenean ibiltzen ikasiko du.

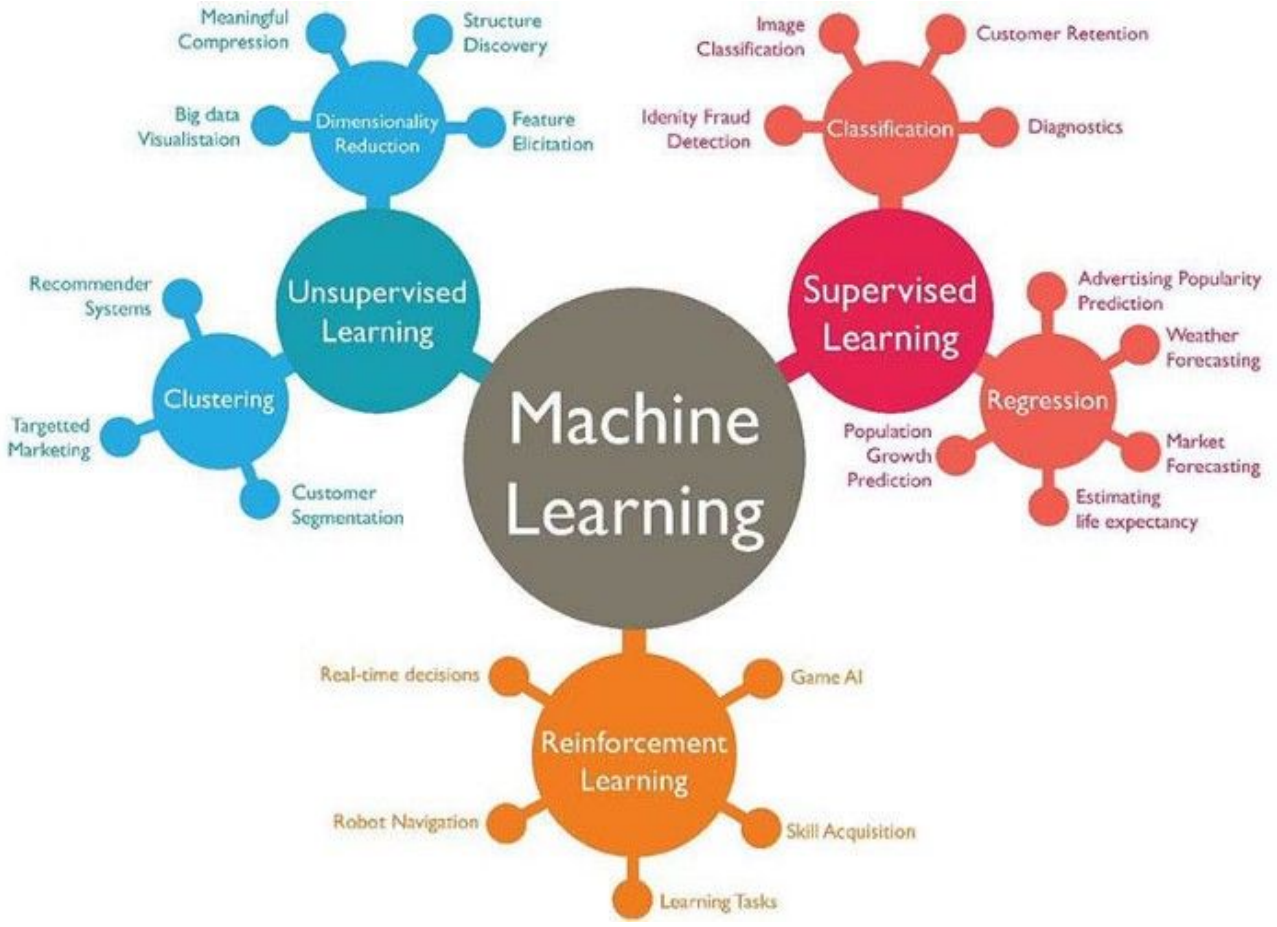


Figure 2. Ikasketa automatiko arloak.

Mario jokoan denborak garrantzi handia du, ingurunea dinamikoa delako lortzen den saria atzerapen batekin datorrelako. Horretarako Mariok zein ekintza segida konkretu egin behar dituen ikasi behar du, sari hau maximizatzeko asmoz. Egoeraren kalitatea erraz neurtu daiteke, baina ekintzak zehaztea zaila da. **Errefortzu bidezko** ikasketak hori lortzeko aukera ona da.

Bideo-jokoen arkitektura oso bateragarria da errefortzu bidezko ikasketarekin, eta ohikoa da bi hauek konbinatzea. Denbora errealeko kasuetan ere erabiltzen dira, non egoera espazioa konplexua den. Hauen adibideak dira robot batek atea nola ireki behar duen ikastea, edo kotxea gidatzen ikastea.

Errefortzu bidezko algoritmoen artean oinarrizkoa den bat erabiliko da: **Q-learning**. Algoritmo hau eskuz inplementatuko da, honi esker kontrol gehiago lortuko da algoritmoaren exekuzioan eta honen funtzionamenduan gehiago sakontzeko aukera egongo da. Honen alternatiba bat **Gym** [6] liburutegia erabiltzea da, errefortzu bidezko algoritmoak ekartzen dituena. Algoritmo honek **Markov erabaki prozesua** erabiltzen du, **Bellman ekuazioarekin** batera balioak eguneratzeko.

Ondoren, algoritmo honen eboluzioa inplementatuko da, **Deep Q-Network** izenez ezagutzen dena. Algoritmo honek q-taula erabili beharrean q-balioen hurbilpena kalkulatzen du, denborarekin emaitza zehatzagoak lortuz. Aurrekoak bezala, Markov erabaki prozesua eta Bellman ekuazioa erabiltzen ditu kontrol moduan.

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

1.1.6 Lanaren Deskonposaketa Egitura eta emangarriak

Proiektuan zehar emangarri hauek sortuko dira:

Plangintza. Proiektuaren ezaugarriak zehazten duen dokumentua.

Dokumentazioa. Bertan proiektuari buruzko informazioa osoa egongo da. Amaieran plangintza dokumentu honetan egongo da ere.

Kode fitxategiak. Lua eta Python lengoaian dauden kode fitxategiak. State fitxategiak ere egongo dira, edozein maila modu errazean kargatzeko emulagailuan.

Aurkezpen dokumentua. Defentsa egiteko erabiliko den fitxategia.

LDE-a bi adarretan banatuko da. Alde batetik proiektuaren programa edo honen kodea egongo da, q-learning eta sare neuronala, Lua eta Python lengoaiak erabiltzen dutenak. Beste aldetik kudeapen eta kontrol adarra egongo da. Bertan planifikazioa, dokumentazioa eta aurkezpen dokumentua egongo dira.

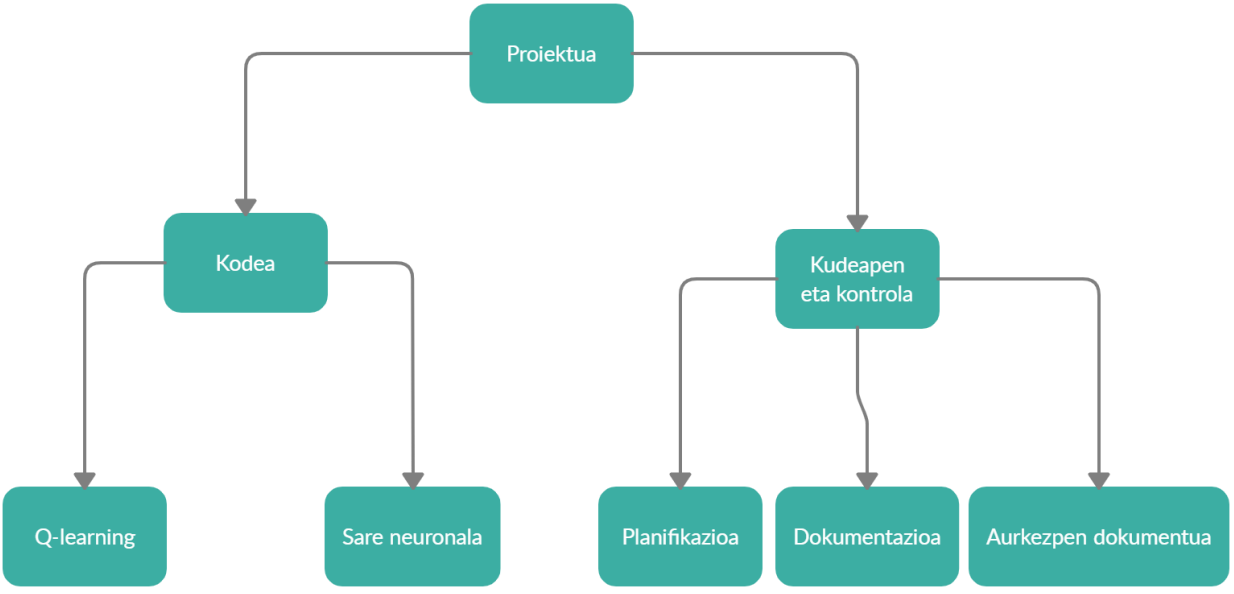


Figure 3. Lanaren Deskonposaketa Egitura

1.2. Atazak egingo diren denboraldiak eta bukatzeko datak

1.2.1 Atazen arteko menpekotasuna

Lehendabizi plangintza egingo da, ondorengo atazak zehazten dituena. Q-learning izango da lehenik egingo den programa, honek Lua erabiliko du. Sare neuronala egiteko beharrezkoa izango da lehenik Lua kode hori izatea, bertan egongo dira eta emulagailua kontrolatzeko behar den kodea, bezero-zerbitzari moduko komunikazio batekin erabili ahal izango duena. Dokumentazioa amaitzeko beste ataza guztiak amaiturik egon beharko dute, proiektu osoaren informazioa hemen egongo da eta. Azkenik, aurkezpen edo defentsa dokumentua sortzeko dokumentazioa beharrezkoa izango da, honen laburpen bat izango delako.

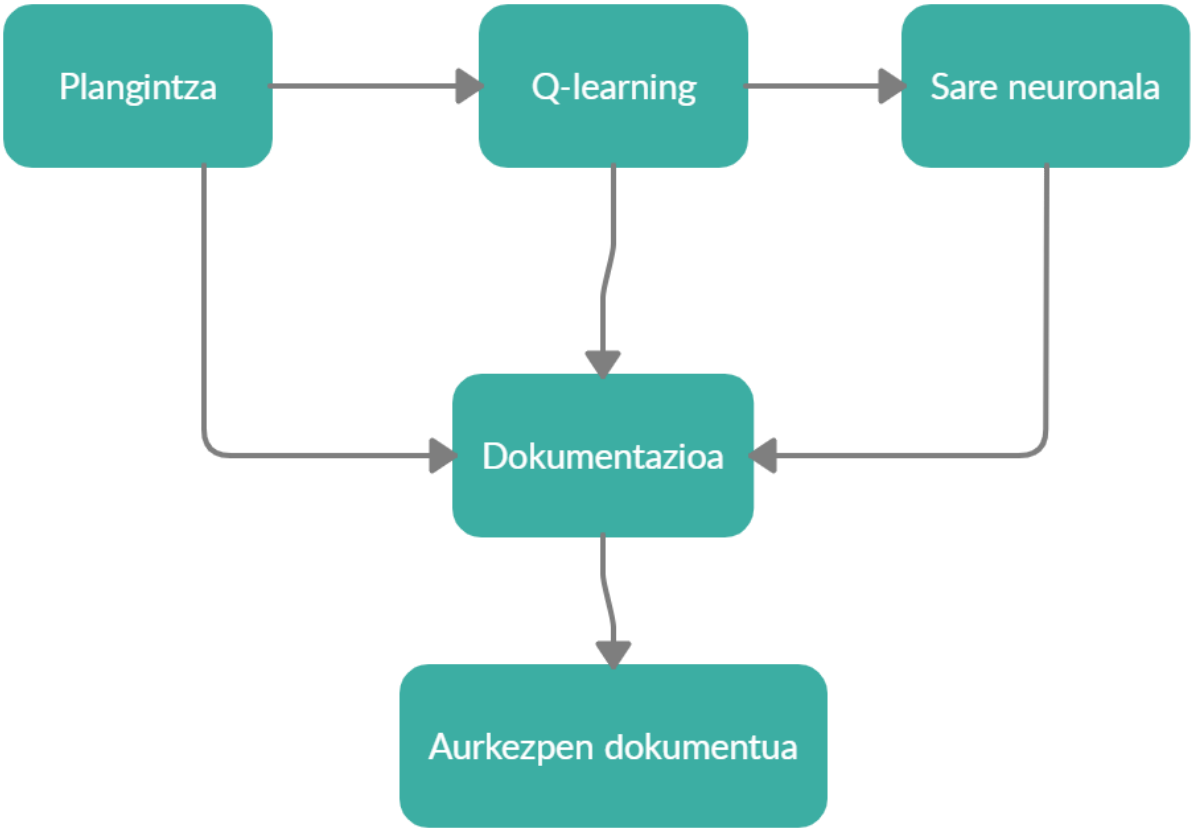


Figure 4. Atazen arteko menpekotasuna

1.2.2 Atazen garapen denboraldiak

Q-learning inplementatzeko hilabete eta erdi espero da eta sare neuronala hilabete batean. Sarearentzat denbora gutxiago behar izatea espero da Lua oinarrizko arkitektura jadanik eginda egongo delako. Dokumentazioa ez da agertzen grafikoan, baina hilabete guztietan ageri dago, etengabeko eguneraketak jasaten.

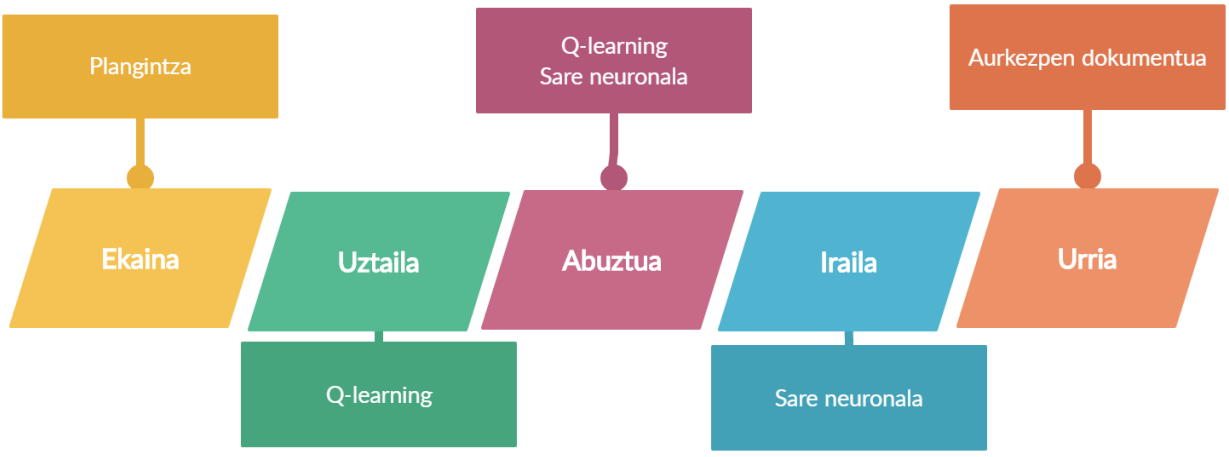


Figure 5. Atazen garapen denboraldiak

1.2.3 Proiektuaren garapen mugarriak

Mugarri garrantzitsuenak dokumentazioarena eta aurkezpen dokumentuarena dira, informatika fakultatearen menpe daudelako. Beste mugarriak pertsonalak dira, proiektuaren exekuzioa kontrolpean izateko.

Plangintza: 2020/06/30.

Q-learning: 2020/08/15.

Sare neuronala: 2020/09/15.

Proiektua ADDI plataformara igo: 2020/10/11.

Aurkezpen dokumentua: 2020/10/19.

1.3 Ataza bakoitzari emango zaion denboraren estimazioa

	Estimazioa (ordu)	Denbora erreala (ordu)
Planifikazioa	10	10
Q-learning	130	145
Sare neuronala	100	135
Dokumentazioa	50	80
Aurkezpen dokumentua	10	-
Guztira	300	370 (+ aurkezpena)

1.3.1 Desbiderapenak

Desbiderapen nagusia sare neuronalean egon da. Programazio arazo batzuk izan nituen, eta sare neuronalak kaxa beltz bat bezalakoa delako zaila da batzuetan arazoa non dagoen aurkitzea. Horretaz gain, sareak denbora handiagoa behar du entrenatzeko (bi ordu gutxienez errore funtzioaren konbergentzia ikusteko) eta denbora behar da hainbat hiperparametroen eta aldaketen eragina ikusteko.

Dokumentazioa egiteko behar den denboraren estimazioa txarra izan da ere, uste baina dedikazio handiagoa behar izan du.

1.4. Proiektuko informazio eta komunikazio sistemen berezitasunak

Proiektuan erabili eta sortuko diren datu guztiak Google Drive karpeta batean gordeko dira.

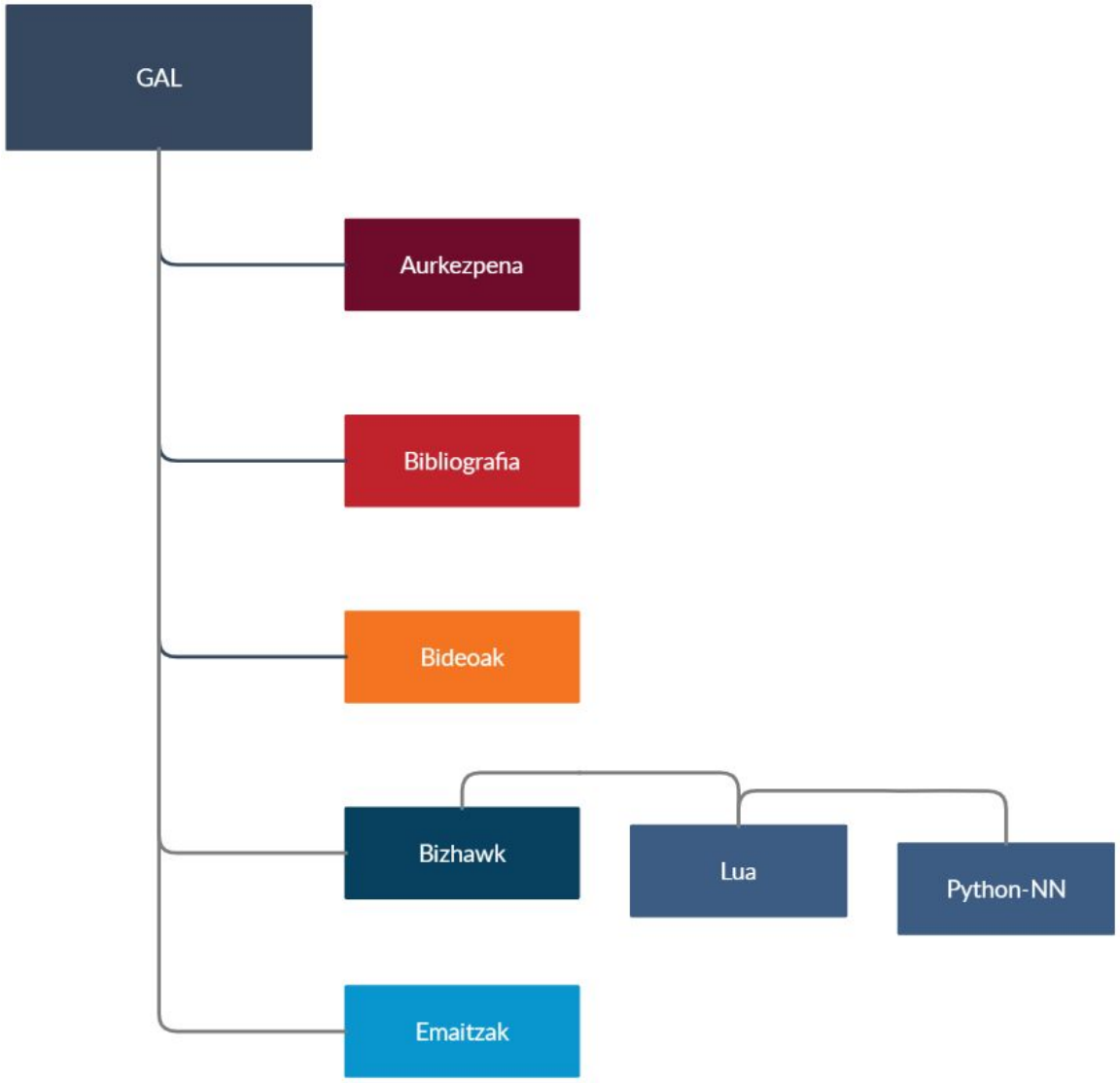


Figure 6. Informazio sistemaren egitura

GAL: Proiektuaren erroa. Dokumentazio fitxategia bertan egongo da, eta beste karpeta guztiak ere.

Aurkezpena: Defentsa egiteko erabiliko diren irudiak eta aurkezpen dokumentua hemen egongo da.

Bibliografia: Ikasteko erabili diren dokumentuak.

Bideoak: Bideoak eta bideo proiektuak hemen gordeko dira.

Bizhawk: Emulagailuaren karpeta nagusia. Bertan exekutagarria egongo da, baita ere jokoen fitxategiak.

Lua: Lua kode fitxategi guztiak eta q-learning inplementatzeko erabili diren Python fitxategiak.

Python-NN: Sare neuronala sortzeko beharrezkoak diren Python fitxategiak.

Emaitzak: Lortutako emaitza grafikoak.

Proiektuaren zuzendariarekin komunikatzeko hiru bide definitu dira. Komunikazio idatzia Telegram zerbitzuaren bidez egingo da. Bilerak egiteko berriz, meet.jit.si erabiliko da, bideokonferentziak egiteko aukera duelako. Larrialdi kasuetan mugikor bidezko deia ere egiteko aukera dago.

1.5. Arriskuen analisia

A1 - Bizhawk emulagailua Windows sisteman exekutatzeko dago egina. Unix sistema batean ere erabili daiteke, baina bug edo arazoak agertzeko aukerak daudenez, Windows erabiliko da. Honek muga bat sor dezake, erabiltzen den tresnaren bat ez bada bateragarria sistema honekin.

Hainbat tresna daudenez aukeratzeko, probablea da Windowsekin bateragarria den bat egotea. Kasurik okerrenean, Unix sistema bat erabili daiteke (makina birtuala edo beste konputagailu batean) eta socket bidez komunikatzea programa.

A2 - Bizhawk emulagailuak Lua erabiltzen du scripting lengoaia bezala, eta oso probablea da erabiliko den tresnaren bat ez izatea bateragarria lengoaia honekin. Kasu honetan Python erabili daiteke eta zubi bat sortu bi lengoaiek elkarrekintza egiteko.

A3 - Lehen aldia denez errefortzu bidezko ikasketa eta sare neuronalak erabiltzen oso probablea da ataza batzuetan esperotakoa baina denbora gehiago behar izatea ikasten.

2. Errefortzu bidezko ikasketa

Sarrera

Errefortzu bidezko ikasketan elkarrekintza bidez helburu zehatz bat lortu nahi dugu. Ikaslea eta erabakiak hartzen dituenari **egile** deituko zaio (**Mario** ere deituko zaio dokumentuan zehar). Egile honek objektu batekin elkarrekintza egiten du, hori **ingurunea** izango da. Egileak ez du ingurunearen informazio osoa, zati bat bakarrik ikusi dezake. Ingurunetik antzematen duen informazioa une konkretu batean **egoera** izango da (jokoaren fotograma baten pixel balioak, adibidez).

Elkarrekintza hau modu konstantean izango da: egileak **ekintza** bat egingo du uneko egoera batean, inguruneak ekintza horri erantzungo dio eta egoera berri bat aurkeztu dio egileari. Baita ere lortu duen **saria**, zenbaki oso bat izango dena.

Mario jokoaren kasuan:

- **Ingurunea** NES kontsolaren emuladorea da
- **Egoera** fotograma bat izango da, informazio grafikoa duena
- **Ekintzak** NES kontrolagailuaren botoi desberdinak eta hauen konbinazioak izango dira (A, B, ezker, eskuin+A…).
- **Egilea** Sorturiko programak kontrolatzen duena izango da, kasu honetan Mario.

Frame bakoitza egoera indibidual bat denez, ingurunea **diskretua** da, eta ez jarraitua.

Ingurunea **guztiz determinista** da, hau da, ez da posible gauza bera eginez emaitza desberdin bat lortzea, hasierako egoera bera izanda. Honen aurkakoa indeterminismoa izango litzateke, adibidez txapon bat jaurtitzea (Bernoulli prozesua).

Errefortzu bidezko ikasketa inplementatzeko **q-learning** erabili da, 1989 urtean sortu zena. Algoritmo honekin egoera bakoitzean zein ekintza egin behar duen ikasiko du sari metatu handiena lortzeko.

2.1 Markov erabaki prozesua

Matematiketan **Markov erabaki prozesua** kontrol prozesu bat da. Modelo matematiko bat sortu behar da egoera batean zein erabaki hartu behar den jakiteko. Egoera horretan emaitzak partzialki ausazkoak izan daitezke eta egile baten kontrolaren menpean daude. Prozesu hau optimizazioak egiteko erabiltzen da. Ekintza eta egoera posibleak finituak direnez, Markov erabaki prozesua finitua dela esan daiteke.

Denbora pausu bakoitzean, prozesua $\mathbf{s_t}$ egoera batean dago. Egileak **a** ekintza posible bat egiten du eta prozesuak $\mathbf{s_{t+1}}$ egoera berria itzultzen dio egileari, baite ere lortutako saria $\mathbf{R(s_t, s_{t+1})}$.

Markov erabaki prozesua 3 elementuz osaturiko tupla baten bidez definitu dezakegu, $\mathbf{(S, A, R_a)}$ non:

S: Egoera espazioa.
A: Ekintza espazioa.
$\mathbf{R_a(s)}$: **s** egoeran **a** ekintza aplikatuta lortzen den saria

Batzuetan laugarren elementu bat erabiltzen da ere: $\mathbf{s_t}$ egoeran **a** ekintza egin ondoren $\mathbf{s_{t+1}}$ egoerara iristeko **probabilitatea**. Jokoa determinista denez, balio hau ez da erabiliko, emaitzak ez direlako partzialki ausazkoak.

$\mathbf{s_{t+1}}$ egoerak duen menpekotasun bakarrak $\mathbf{s_t}$ eta honi aplikaturiko **a** ekintza dira. Honek iraganak ez duela menpekotasunik sortzen orainaldian esan nahi du, edo memoria ez duela. Hau **Markov propietatea** da, eta Markov Erabaki Prozesuen egoera trantsizioak propietate hori betetzen du.

Honek galdera bat sortzen du, Super Mario jokoaren inguruneak Markov propietatea betetzen al du? Algoritmoak frame bakoitza pixel moduan jasoko du, posiblea da hurrengo egoera zein izango den jakitea, uneko egoera kontuan izanda bakarrik?

Figure 7. Mario eta Markov propietatea

Erantzuna ezezkoa da, ezin da jakin zein izango den hurrengo egoera. Honen arrazoia irudia estatikoa delako da, irudi bakar batekin ezin da jakin Marioren abiadura edo norabidea. Iraganak menpekotasuna sortzen du, informazioa falta da Mario erortzen dagoen ala oraindik saltoa egiten dagoen jakiteko.

Honi soluzioa emateko hurrengoa proposatzen da: egoera frame bat izan beharrean, **lau frame** ordenatuta izan daitezke. Honi esker egoera bakoitzean abiadura eta norabidearen informazioa gordeko da modu inplizituan. Markov propietatea indartsuagoa izango da horregatik.

Horretaz gain, Mariok salto egin ondoren salto egiteko botoia mantentzen badu, inoiz askatu gabe, **ezingo du salto egin** berriro, blokeo bat sortzen delako. Lehenik salto egiteko botoia askatu behar du, blokeoa askatzeko eta ondoren salto egiteko. Honek iraganarekiko menpekotasuna sortzen du, ezin delako guztiz jakin Mario lurrean dagoenean salto egiteko botoia zapaltzen duenean zer pasako den, eta honek etengabeko ziklo bat sor dezake egoera konkretu batzuetan.

Hau konpontzeko falta den informazioa irudian kodifikatu da. Azken ekintza “salto” izan bada, ertz bateko 2x2 pixel zuriz margotu dira, eta bestela beltzez. Horrela Mariok kontrol gehiago izango du jauzi egiteko garaian.

Azkenik, pantailak 256 pixel baditu, eskuinetik etorriko diren hurrengo 256 pixel aurretik kargatzen ditu, baita ere hemen agertzen diren etsaiak. Honen ondorioz ingurunearen errepresentazio grafikoa ez da totala izango, beti faltako delako pantailatik kanpo dagoen zatia. Honen ondorioz egoera batzuetan Markov propietatea ez da betetzen, eta honek ikasketa geldotu dezake.

2.2 Sariaren definizioa

Saria kalkulatzeko funtzio bat sortu behar da, egoera bakoitza neurtzen duena. Egoera ona bada, saria altua izango da. Hau definitzeko Mario mugitu den distantzia izango da. Joko honetan mailaren amaiera eskuinean dagoenez, eskuinera mugitzen den eñean egoera hobeagoa izango da.

Mario x_1 posizioan badago eta hurrengo egoeran x_2 posizioan, lortu duen saria x_2-x_1 izango da, hau da, Marioren abiadura. Metatutako saria berriz, une horretan Mariok duen posizioa x ardatzean.

Pasatako denbora penalizazio moduan erabiltzea ere pentsatu da, baina azkenean ez erabiltzea erabaki da. Penalizazioak erabiltzen badira sari metatuak ez du Marioren x posizio zehatza gordeko, eta honek inplementatu diren beste sistemekin arazoak sor ditzake (ekintza distantzia kalkulatzerako garaian). Saria beti 0 edo handiagoa denez, informazio hau kasu batzuetan erabilgarria izango da.

Mariok joko barruan puntuak irabazten ditu etsaiak hiltzean, txanponak hartzean, eta besta hainbat gauza eginda. Honek maila bat pasatzerako garaian garrantzi handirik ez duenez, baztertua izan da.

2.3 Q-taula

Markov erabaki prozesuaren informazioa hiru zutabedun matrize batean gorde daiteke, **(s,a,R_a)** itxura duena (egoera, ekintza eta lortuko den sari metatua). Q-taula izenez ezagutuko da datu egitura hau, eta bertan gordeko da ikasitako informazioa.

Algoritmo klasikoan q-taularen balio posible guztiak hasieratu egin behar dira programa abiaraztean, baina gure kasuan hori ezinezkoa da. Egoera batek 256 bit okupatzen baditu, (ekintzaKopurua) * 1.2 x 10^77 egoera desberdin hasieratu beharko litzateke, beraz tamaina dinamikoa duen egitura bat erabili behar da.

Hiztegi / hash taula bat erabiliko da informazioa gordetzeko. Hasieran hutsa egongo da, eta egoera berriak topatzerako garaian hiztegian txertatuko da.

Hiztegiaren indizeak egoera eta ekintzak izango dira, horrela atzipena O(1) denbora konplexutasunean egingo da eta erabilpena oso erosoa da. Gordetzen duen balioa **s** egoeran **a** ekintza aplikatuta lor dezakeen sari maximoa da, etorkizun urrunera begira, eta balio hauek Bellman ekuazioa erabiliz kalkulatuko dira. Q-taulak itxura hau izango luke:

egoera \ ekintza	Salto egin	Eskuinera mugitu	Eskuinera korrika mugitu	Ezkerrera mugitu
s_0	0	2	0	0
s_1	1	2	0	0
...	...	...	...	

2.4 Bellman ekuazioa

Bellman ekuazio bat erabiliko da q-taularen baloreak eguneratzeko. Honek lortuko den uneko sariaren eta etorkizunean lortuko den sari totalaren batezbesteko pisudun bat kalkulatzen du. Programazio dinamikoan erabiltzen da ekuazioa hau, arazo handi bat arazo txikietan banatzen denean, soluzio sub-optimo edo optimoak aurkitzeko asmoz.

Ekuazio honekin Mariok zer ekintza egin behar dituen ikasiko du, sari maximoa lortzeko asmoz partida amaieran.

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \Big)}^{\text{temporal difference}} - \underbrace{Q(s_t, a_t)}_{\text{old value}}$$

$$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{new value (temporal difference target)}}$$

$Q^{new}(s_t, a_t)$: Q-taulak izango duen balio berria [s_t, a_t] posizioan .
$Q(s_t,a_t)$: Posizio horretan aurretik zegoen balioa.
r_t: **s_t** egoeratik **s_{t+1}** egoerara igarotzean lortzen den saria.
$maxQ_a(s_{t+1},a)$: Hurrengo egoeran lor dezakeen balio maximoa, ekintza posible guztiak kontuan edukiz. Dei errekurtsibo bat da hau, s_t egoera s_{t+1}ren menpe dagoelako, eta hau s_{t+2}ren menpe, amaierako egoerara iritsi arte.

Ekuazioak bi hiperparametro erabiltzen ditu, non [0, 1] tarteko balioak hartzen dituzten:

α: Ikasketa ratioa. Balio hau 0 bada ez du ezer berririk ikasiko, eta 1 bada lortutako informazio berria bakarrik erabiliko da. Ingurunea guztiz determinista denean, gure kasuan bezala, balio hau 1 izatea gomendatzen da.

γ: 0 denean, uneko sariak bakarrik izango ditu kontuan (bista motza), eta 1 denean etorkizunean lor dezakeen sari handiena lortzea izango du helburu. Kasu honetan 1 izango da, Mariok etorkizunean lortuko duen saria maximizatzea delako helburua.

Hiperparametroek balio horiek hartzen dituztenean (1 eta 1), ekuazioak forma hau hartzen du.

$$Q^{new}(s_t, a_t) \leftarrow \quad r_t \quad + \max_a Q(s_{t+1}, a)$$

Algoritmo hau aplikatzean arazo bat azaldu da: deia errekurtsiboa denez uneko egoerak ezin dira berritu partida amaitu arte. Errekurtsibitatea atzeraka egin daiteke ere, egoera bakoitzean aurreko egoera guztiak berritu hasierako egoerara iritsi arte, baina ez da efizientea egoera bakoitzean hau egitea. Partidak n egoera baditu, guztira 1+2+3+4+...+n = $\frac{n(n+1)}{2}$ aldiz egin behar da eguneraketa.

Honi soluzioa emateko stack batean gordetzen dira partida horretako egoera / ekintza bikoteak eta partida amaieran errekurtsioa simulatzen da q-taula eguneratzeko. Hasiera batean sarien stack bat ere sortu zen, baina Bellman ekuazioa aldatuz ez da beharrezkoa informazio hau gordetzea, memoria espazioa irabaziz. Azken ekuazioan ikusten den moduan, $Q^{new}(s_t,a_t) = R_f - R_t$ betetzen da, non R_f amaierako egoeran lortutako sari metatua den, eta R_t uneko egoerak duen sari metatua. Hau da, oraindik lortu dezakeen sari kopurua. Egoera bakoitzean R_t sari metatua konstantea denez, posiblea da $Q^{new}(s_t,a_t) = R_f - R_t + R_t = R_f$ erabiltzea. Horrela, n egoerako partida bat amaitu ondoren gehienez n balore eguneratzen dira q-taulan, eta ez da beharrezkoa izango egoera bakoitzaren sari metatua gordetzea. Emaitza bera izango da, egoera bakoitzaren balio maximoak ez direlako aldatzen, baina beste ikuspuntu bat erabiliz. Q-balio honi uneko sari metatua kenduz ekuazio originalaren balioa lortu daiteke modu errazean ere.

Horrela, Mariok egoera bakoitzean badaki ekintza bakoitzarekin lortuko duen sari maximoa, une bakoitzean ekintza hoberena egiten badu.

2.5 Politika

Mariok zein ekintza egin behar duen aukeratzeko balio bidezkoa izango da, etorkizun urrunean lortuko duen saria maximizatu nahian. Ekintza hoberena nola aukeratu erabaki behar da. Mariok edozein ekintza egiten du egoera ezezagun batean dagoenean, eta q-taula betetzen doa egileak lorturiko informazioarekin hurrengo partidetan erabiltzeko.

Aukera bat beti ekintza hoberena egitea da, baina honek arazo bat du: Mariok ez luke ezer berririk ikasiko, ez dituelako gauza berriak probatuko. Demagun Mario lehen etsaia ukituz hil egiten dela, hurrengo saiakeran gauza berdina egingo luke, etsai horrekin hiltzean sari maximoa lortu duela ikasi duelako. **Ekintza hoberena egitea ez da beti hoberena izango**, q-taulak ez duelako informazio osoa. Mariok gauza berriak aztertu behar ditu azken partidako oztopoei aurre egiteko.

Aukera bat portzentai bat erabiltzea da, eta horren arabera ausazko ekintza bat egitea. 99% probabilitatearekin ekintza hoberena egingo du, eta 1% ausazkoa den bat (ε-greedy izenez ezagutzen da hau). Honek beste arazo bat sortzen du, egoera kopurua handia delako partida batean. Demagun Mario 2000 pixel mugitzea lortzen duela, baina zulotik behera erortzen dela partida amaituz. Hurrengo partidan gutxienez 2000 pixel mugitzea lortu nahi du, baina oso probablea da hasierako egoera batean ausazko ekintza bat egitea, eta honen ondorioz pentsatuta zuen bidetik ateratzea eta guztiz galtzea. Egoera berri batean sartu da eta ez daki nola itzuli sari maximora hurbiltzeko egin behar dituen ekintzak. Adarkatzea handiegia da ε-greedy erabiltzen bada, eta ikasitako informazioa ez du ondo aprobetxatzen.

Arazo honi aurre egiteko soluzio misto bat hartzea erabaki da, eta hau egitea posiblea da sariaren definizioa Marioren posizioa delako, penalizazio eta sari estrarik gabe. Ekintza hoberena egingo du hasieran (**segurtasun distantzia**), eta uneko sari metatua gerturatzen doan eñean sari metatu maximora ausazko ekintza bat egingo du probabilitate baten menpe (**ekintza distantzia,** edo arrisku distantzia). Hau da, adarkatu aurretik sakondu egiten du. Honekin azken aldian pasa ez zuen oztopo bat pasatzeko ekintza berriak probatuko ditu, eta modu eraginkor batean erabiliko du ikasitako informazioa. Beraz, segurtasun distantzia jadanik ikasi duen hasierako bidea izango da, eta ekintza distantzia barruan sartzen denean ausazko ekintzak egiten hasiko da probabilitate baten menpe. Aurreko adibideari begiratuz eta ekintza distantzia 200 bezala definitzen bada, ekintza hoberena egingo luke 1800ko saria lortu arte, eta ondoren gauza berriak probatzen hasi emaitza hobetzeko asmoz.

Distantzia hau definitzeko garaian pantailaren tamaina erabili da. Pantailak 256 pixel baditu, zentzua dauka horren inguruko balio bat erabiltzea, suposatuko da Mariok egin duen ekintza txarra distantzia horren barruan egongo dela. Honek muga bat suposatu dezake beste joko edo ingurune batzuetan, ekintza baten eragina ikusteko epea oso luzea bada.

Probabilitatea definitzeko bi aldagai definitu dira: maxgreed eta greeddiff. maxgreed aldagaiak ekintza distantzian ekintza hoberena aukeratzeko probabilitate maximoa gordeko du. greeddiff aldagaiak berriz, probabilitate diferentzia gordeko du, zenbat jaistea nahi den. Bi balio hauek erabiliz zuzen baten ekuazioa sor daiteke, eta tarteko balioak interpolatu.

Segurtasun distantzian ekintza hoberena aukeratzen du beti, eta ekintza distantzian ausazko ekintza egiteko probabilitatea handitzen doa, gauza berriak ikasteko asmoarekin.

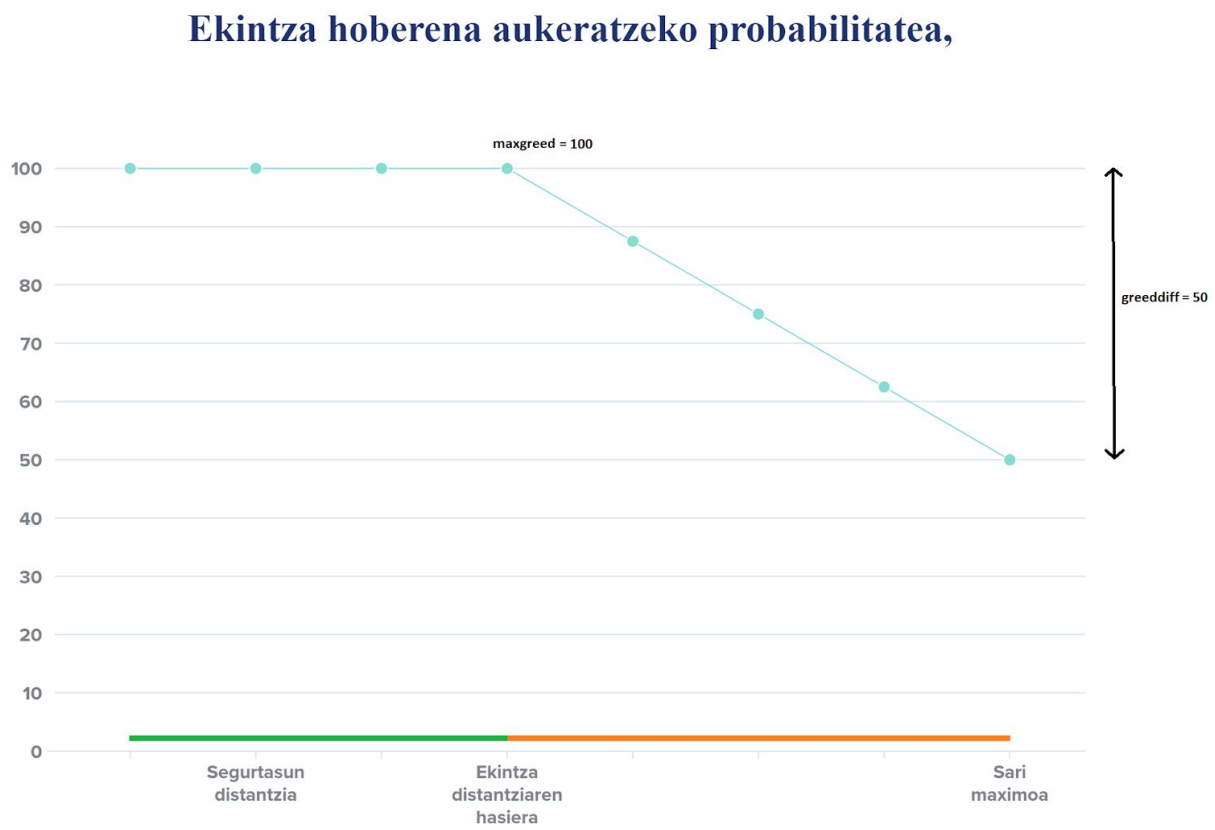


Figure 8. ε-greedy balioak distantziaren arabera

Ikusten den moduan, segurtasun distantzian ez du ezer berririk ikasten. Mariok denbora asko galtzen du segurtasun distantzia garraiatzen, baina posiblea da savestate bat sortzea segurtasun distantzia amaiera inguruan, maila hasieratik hasi beharrean azken oztopoaren aurretik dagoen egoera batean hasteko. Honek denbora asko aurrezten du eta ikasketa denbora handitzen du, maila luzeetan batez ere, non amaieran oztopo zailak dituzten. Aldaketa hau eginez gainera, denbora konplexutasuna $O(n^2)$tik $O(n)$ra jaisten da.

Ekintza maiztasuna ere definitu behar da. Frame bakoitzean ekintza bat egin beharrean, aukeraturiko ekintza hurrengo lau frametan errepikatuko da. Horrela konputazio kostua jaisten da algoritmoa azkartuz. Pertsona normal batek horrela jolasten du ere, ez du frame bakoitzean ekintza desberdin bat egiten.

2.6 Egoeraren definizioa

Algoritmoarentzat egoera nolakoa den definitu behar da. Programak horrela hautemango du jokoan pasatzen ari dena.

Aukera posible bat frame bakoitzean jokoaren RAM memoria osoa irakurtzea izan daiteke, baina hau baztertua izango da. Zentzua dauka programak ikusten duen egoera pertsona normal batek ikusten duenaren antzekoa izatea.

Hasierako inplementazioan oraindik Python komunikazioa ezarri gabe zegoen eta ez zen posible pantailaren argazkia erabiltzea, horregatik lehendabizi tile informazioa erabili da. Tile bat 8x8 pixel tamaina duen elementua da. Tile informazio hau RAM balio zehatz batzuk irakurriz lor daiteke (CIRAM izeneko memorian), eta guztira 32x30 tamainako irudi bat sortu. Hau eginez zehaztasuna galtzen da, eta horrekin **Markov propietatea**. Horren ondorioz, algoritmoak egoerak nahastu egiten zituen batzuetan eta begizta batean sartzeko aukera dago, ikasketa guztiz geldituz. Eranskinak atalean dago azalduta nola egin den hau.

Figure 9. Tile balioak RAM bidez

Horrela ikusiko litzateke 32x30 irudiko jokoaren Tile errepresentazioa.

NES kontsolaren bereizmena 256x224 pixel tamainakoa da, eta hau da egileak ikusiko duen egoera. Pixel bakoitzaren balioa 0-255 tartean definituriko zenbaki oso bat izango da. Horretarako gris eskala erabiliko da, RGB balioen batezbestekoa kalkulatuz. Honek egoeraren dimentsioa jaisten du, datu gutxiago okupatuz memorian. Hau ez da guztiz beharrezkoa q-learning algoritmoarekin, sha balioak kalkulatuko direlako datu tamaina murrizteko, baina sare neuronalean oso erabilgarria izango da.

Ez da lortu irudia Lua bidez atzitzea, beraz Python programari eskakizuna egin eta honek http mezu bidez itzultzen du hau. Pixel batek 256 balio desberdin izan ditzake, beraz byte bat okupatuko du bakoitzak. Guztira 256 x 224 = 57344 byte dira. Mezuaren tamaina jaisteko eta komunikazioa azkartzeko SHA-256 funtzioa aplikatuko da irudian. Hau egin ondoren egoera bakoitzak 256 bit / 32 byte okupatuko ditu, 1792 aldiz gutxiago. Honek ere espazio gutxiago okupatuko da stack-ean gordetzen denean, errekurtsioa simulatzeko. Segurtasun aldetik, hash talka bat gertatzeko probabilitatea ez da 0%, baina oso hurbil dabil. 256 biteko aldagai batek 1.2 x 10^77 balio desberdin posible izan ditzake. Talka hau gertatzeko probabilitatea 0.000001% izatea nahi badugu 4.8 x 10^35 egoera desberdin erabili behar dira (oso balio handia dena).

Izan dezakegun egoera kopuru maximoaren hurbilpen bat egin daiteke. Suposa dezagun segundoero 150 frame prozesatzen direla eta egoera guztiak desberdinak direla. Oso zenbaki eskuzabalak dira hauek, normalean 100fps-ra ez delako iristen entrenamendu saioan eta egoera gehienak errepikatu egiten direlako. Gehienez q-taulan 150 * 3600 = 5.4 * 10^5 egoera desberdin egotea espero da.

Aurreko 4.8 x 10^35 baliora hurbildu nahiko bagenu, talka gertatzeko aukera txiki hori izateko, programa 10^26 urte utzi beharko genuke exekutatzen. Konparazio bat egiteko, unibertsoak 1.4 x 10^10 urte ditu.

Beraz, talka gertatzeko aukera baztertu daiteke. (ikus urtebetetzearen paradoxa[7] edo urtebetetze erasoa[8] informazio gehiagorako).

Egoera espazioa sha balio hauek osatuko dute q-learning algoritmoan.

2.7 Ekintzen aukeraketa

Mariok ingurunearekin komunikatzeko egingo dituen ekintzak definitu behar dira. Software bidez NES kontroladore baten botoiak zapalduko dira, eta honek zortzi botoi ditu guztira: A, B, ezkerra, eskuin, gora, behera, start eta select.

Figure 10. NES kontrolagailua

[7] https://en.wikipedia.org/wiki/Birthday_problem
[8] https://en.wikipedia.org/wiki/Birthday_attack

Ekintza bat botoi bat edo botoi konbinazio baten zapalketa izango da (adibidez A+B, eskuin+A, eskuin+gora+A+B…).

Mandoak fisikoki muga bakarra du: gora eta behera ezin dira aldi berean zapaldu, eta eskuin eta ezker ere ez. Posiblea da software bidez hori egitea, baina kontuan eduki behar da mando normal batekin ezin dela hori egin.

Botoi guztiak eta konbinazio guztiak erabiliz gero, ekintza multzoa oso handia izango da, eta gainera hainbat botoi ez dira erabilgarriak. Horregatik ekintza multzo bat aukeratu behar da.

Demagun Select eta Start ez direla erabiltzen, eta beste botoien konbinazioak erabiltzen direla, non gehienez 3 botoi zapaldu ditzakegun.
$C(n,\ r) = \frac{n!}{(r!(n-r)!)}$ ekuazioa erabiliz konbinazio posibleak kalkulatu daitezke.
C(6, 1) + C(6, 2) + C(6, 3) = 6 + 15 + 20 = 41. Ekintza hutsa eta mandoaren muga fisikoak kontuan edukiz, 40 ekintza posible edukiko genituzke.

Honek ikasketa asko geldotzen du, eta botoi guztiak erabili ordez, ideia ona dela dirudi ekintzak mugatzea.

Ezer ez: Ez zapaldu botoirik. Mariok itxaron egingo du, ezer berririk egin gabe.

Start: Jokoa eteten du. Beste botoiak desgaitzen ditu, berriz Start zapaldu arte.

Select: Ez du ezer egiten.

Gora: Ez du ezer egiten.

Behera: Pipe batzuetan sartzeko erabiltzen da, eta Mario handia bada makurtzeko erabiltzen da. Botoi hau ez da erabiliko, jokoan inpaktua oso txikia delako eta ikasketa geldotzen duelako. Botoi hau kentzen bada soluzio optimo batzuk ez dira aurkituko.

B: Botoi hau zapaltzerakoan Mario korrika hasiko da. Bakarrik zapaltzerakoan ez du ezer egiten, beraz B botoia ezker edo eskuin batekin erabiliko da gutxienez.

A: Mariok salto egiten du hau zapaltzerakoan. Botoia zapalduta mantentzen bada, saltoa handiagoa izango da. A bakarrik zapaldu daiteke, eta alboetara salto egiteko konbinazioak ere erabiliko dira (A+B+eskuin/ezker).

Eskuin: Mario eskuinera mugituko da. Botoi hau B botoiarekin ere erabiliko da, abiadura azkartzeko.

Ezker: Mario ezkerrera mugituko da. Botoi hau oso interesgarria da, berezitasun bat du: saria txikitzen du eta helburutik urruntzen da. Maila guztietan Marioren helburua eskuinera joatea da, maila amaiera hemen dagoelako, eta ezker botoiak ikasketa prozesua geldotu dezake, egoera askotan ekintza hori ez delako izango ekintza optimoa.

Posiblea da ezkerra ez erabiltzea eta arazorik ez izatea ikasketan? NES kontsola zaharra denez, garai haietan muga gehiago zituzten jokoak diseinatzeko garaian. Horren ondorioz, Marioren maila guztiak horizontalak dira, ez dago bertikala den mailarik. Gainera, mailak egiten duten scrolling-a eskuinera mugitzen da bakarrik. Kamara eskuinera mugitzen denean, ezin da atzeraka bueltatu. Honen ondorioz, suposatu daiteke mailak diseinatzerakoan kontuan izan zutela muga hori (scrolling horizontala ezkerrera ezin dela egin), eta botoi honek joko honetan duen garrantzia ez dela hain handia mailak pasatzerako unean.

Figure 11. Mario erabaki desegokia hartzen

Irudian ikusten den moduan, Mariok ezin badu ezkerra erabili blokeatuta geratuko da. Garrantzitsua da ikustea beheran badagoela bidea, Mariok salto egin ez balu aurrera jarraituko luke, eta ez litzateke beharrezkoa izango ezkerrera mugitzeko botoia zapaltzea.

Ekintza multzo hau erabiliko da proiektuan:

A = { A , eskuin, eskuin+B, eskuin+A+B }.

Mariok 50% probabilitatea izango du salto egiteko, eta saltoa goraka ala eskuinera izango da. Gainera Mariok bere abiadura aldatzeko hainbat aukera izango ditu ere.

Emaitza optimoak aurkitzeko aukerak galtzen dira, baina botoi horiek erabiliz Mariok ez luke arazo handirik izan beharko soluzio azpi-optimoak aurkitzeko mailak pasatzerako garaian. Ikasketa asko azkartuko du ere, inpaktu erreala duten ekintzak egiten saiatuko delako, erabilgarriak ez direnak baztertuz. Beraz Mario eskuinera mugitu ahal izango da bere abiadura aldatuz, eta salto ere egingo du.

2.8 Amaierako egoera

Garrantzitsua da Mariok partida noiz amaitzen duen detektatzea, algoritmoaren kontrola ona izan dadin.

Mariok partida bat bi kasuetan amaitzen du:

Lehen kasua, Mario hiltzen denean (etsai bat ukitzean edo zulotik behera erortzean adibidez). Kasu hau RAM memoria atzituz jakin daiteke. Mario zulotik behera erortzean segundo batzuk pasatzen dira partida amaitu arte, baina RAM memoria bidez unean detektatu dezakegu hau (Marioren Y posizioa negatiboa denean) partida berriz hasteko segundo horiek galdu gabe. Mailaren denbora amaitzen bada Mario hil egingo da ere.

Ikasketa azeleratzeko bigarren kasu bat sortu da: Mariok ez badu bere saria denbora finitu batean handitzen, partida berriz hasiko da. Hau erabilgarria da Mario trabatuta geratzen den kasuetarako, 400 segundoak pasa arte itxaron beharrean unean hasiz partida berria ekintza efizienteago batzuk bilatuz. Honek emaitza ez aurkitzea saihesten du ere, posiblea delako Mariok denbora asko galtzea maila baten hainbat puntuetan, eta honen ondorioz ezin izatea amaierara heltzea denbora faltagatik.

Amaierako egoera batera iristean, partidan zehar lortutako informazioa erabiliz q-taula berrituko da. Errekurtsibitatea erabili beharrean begizta bat erabili da balioak berritzeko. Partida horretako **s** egoera bakoitzaren $\mathbf{maxQ_a(s_{t+1},a)}$ balioa kalkulatuko da, hau da, ekintza posible guztiak eginda lor dezakeen sari maximoa. Ondoren balio hau partidan lortu den amaierako sariarekin konparatzen da. Lortutako sari berria altuagoa bada, **s / a** egoera/ekintza berritu egiten da q-taulan.

2.9 Algoritmoaren logika eta konplexutasuna

Algoritmoak nola funtzionatzen duen azalduko da atal honetan, egiten diren pauso guztiak jarraituz:

1. Aldagaiak hasieratu eta q-taula hutsa sortu.
2. Programaren begizta nagusian sartu, while(true) bat izango dena. Begiztaren barruan hurrengo ekintzak egingo dira:
3. Emulagailuari hurrengo egoerara pasatzeko eskatu, aukeraturiko ekintzarekin.
4. Uneko egoera lortu. Egoera ez badago q-taulan, balio hau hasieratu, aurrerago bertan idazteko.
5. Mariok partida amaitu duen ikusi. Hau gertatu bada, partida horretan ikasi duenarekin q-taula eguneratu Bellman ekuazioa erabiliz, eta partida berriz hasiko da ondoren. Egoera eta ekintzen stack-ak hustu egingo dira partida berrirako.
6. Uneko egoeraren ekintza hoberena zein izango den begiratu taulan. Ekintza distantzia barruan badago, probabilitate bat egongo da ekintza hau ausazkoa izateko.
7. Hurrengo egoerara pasatzerakoan erabili behar den ekintza gorde.
8. Uneko egoera eta aukeratu den ekintza stack batean gorde, partida amaitzerakoan q-taula eguneratzeko.

9. Hirugarren pausora bueltatu, begiztaren hasierara.

Denbora konplexutasuna zehazteko kasurik okerrena zein den kalkulatuko da. Demagun Mariok pasa behar duen mailaren distantzia N dela. Kasurik okerrenean, Mariok partida bakoitzean pixel bat mugitzea lortuko du. Lehen partidan lortutako saria 1 izango litzateke, bigarrenean 2, ondoren 3… eta horrela N-ra iritsi arte. Mariok 1+2+3+...+N partida behar baditu maila pasatzeko, algoritmoaren denbora konplexutasuna $O(n^2)$ izango litzateke. Hau ez da gure kasua, segurtasun distantziaren amaieran checkpoint bat jartzen delako ikasketa azeleratzeko asmoarekin. Ekintza distantzia 100 bada, Mariok partida bakoitzean 101 ekintza egin beharko ditu pixel bat aurreratzeko. Teknika honi esker 101*N partida beharko ditu maila pasatzeko, denbora konplexutasuna O(n)-ra jaitsiz.

2.10 Lua eta Python arteko komunikazioa

Bi lengoaien arteko zubia sortzeko hainbat aukera probatu dira:

- **Pipe** bat sortu, Lua bidez cmd komando bat exekutatu daiteke (Windows Command Prompt). Honekin “python_programa.py” Python script bat exekutatu daiteke eta honek print() bidez idazten duen informazioa atzitu. Aukera hau oso geldoa da, Python frame bakoitzean kargatzen delako, eta liburutegi batzuek denbora behar dute hasieratzeko.

- **Socket** komunikazioa erabili. Honen arazoa sinkronizazio falta izan da. Socketaren emaitza lortzeko itxaron denbora definitu behar da, eta funtzioa denbora hori pasa arte itxaroten geratzen zen, emaitza lortzeko. Honek bi arazo sortzen zituen: batzuetan erantzun denbora azkarragoa zen, beraz denbora galtzen zen itxaroten. Beste aldetik, emaitza denbora geldoegia bazen, ez zuen daturik lortzen eta berriro saiatu behar zen.

- **HTTP**. Python bidez http zerbitzari bat sortzea izan da emaitza hoberena. Mezuen sinkronizazioa ona izan da eta emaitza denbora azkarra. Lua zerbitzariak JSON formatuko string bat eraiki ondoren POST bidez egingo da eskaria. Pythonek informazioa atzitu eta egin beharreko operazioak aplikatu ondoren emaitza bueltatuko du.

Beste komunikazio mota bat erabili da ere: **clipboard**-a (ctrl+c edo “Copiar” egitean sistemak erabiltzen duen memoria). Emuhawk emuladoreari esker Lua funtzio baten bidez irudia Clipboard-era pasatzeko aukera dago. Honekin Pythonek zuzenean atzitu dezake irudia, datu gutxiago erabiliz POST mezuan eta fitxategia diskoan idaztea ez da beharrezkoa. Honen desabantaila zerbitzaria eta jokoa makina berean exekutatu behar direla da, baina beste aukera batzuk ere badaude (PNG fitxategi moduan gorde, edo http post mezuan bidali).

2.11 Aurkitutako zailtasunak eta soluzio posibleak

Inplementatu den algoritmoak zailtasunak izan ditu kasu konkretu batzuetan, ikasketa geldotzen edo trabatzen dutenak.

Ikasketa geldotzen dutenak:

Definituriko ekintza multzoa begiratuta (A, eskuin, eskuin+B, eskuin+A+B) saltatzeko probabilitatea 50%koa dela ikus dezakegu. Egoera berrietan aukeratutako ekintza ausazkoa denez (ekiprobableak izanda lau ekintzak) Mariori kosta egiten zaio salto luzeak egiteak, oso beharrezkoak direnak oztopo mugari gainditzeko. Salto luze bat egiteko A edo eskuin+B+A ekintzak hainbat alditan egin behar ditu segida batean, eta ausazkoak direnez ekintzen joera salto txikiak egitea da. Honek ikasketa zati batzuetan geldotzen du, soluzio bakarra salto luzea egitea bada.

Jokoan bi ur maila daude (2-2 eta 7-2), eta grabitatea guztiz aldatzen da maila hauetan. Aurreko arazoa maila hauetan nabariagoa da, Mariok A botoia zapaltzen duenean gora igotzen da, eta ondoren denbora behar du behera jaisteko, oso abiadura geldoan jaisten delako. Maila hauetan Mario goran egoteko joera zuen, ausazko ekintzen ondorioz, hainbat frame-etan A ez duelako zapaldu behar behera jaisteko. Honen ondorioz Mariok denbora gehiegi behar zuen berez errazak diren oztopoak gainditzeko.

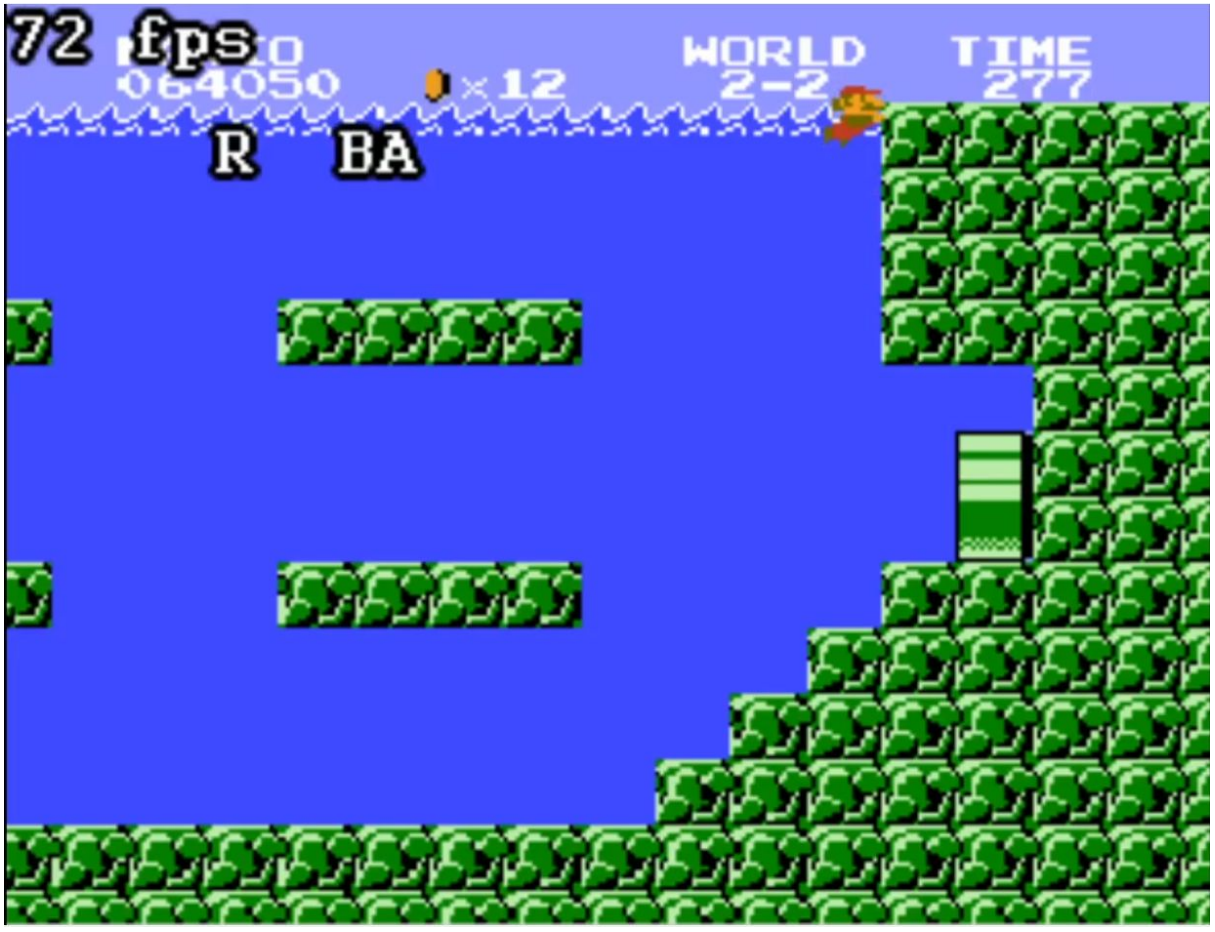


Figure 12. Mariok zailtasunak ditu jaisteko

Mariok egin behar duen gauza bakarra azpian dagoen pipe horretan sartzea da, baina denbora asko behar zuen hori pasatzeko.

Arazo honi aurre egiteko ekintzen probabilitatea aldatzea izan da. Ez da beharrezkoa izan salto luzeak egiteko probabilitatea aldatzea, ez duelako hainbeste denbora behar hori ikasteko, baina ur mailetan aldaketa hau aplikatu ondoren Mariok kontrol gehiago lortu du igeri egitean. Probabilitate hauek erabili dira:

A: 10%
A+B+eskuin: 10%
B+eskuin: 40%
eskuin: 40%

Segurtasun distantziak egoera batzuetan arazoak sortzen ditu ere. Maila batzuetan ez dago lurra, eta eta ohikoa da salto luze bat egin ondoren Mario zulora erortzea eta partida amaitzea. Maila hauetan lortzen den saria atzerapen handiago batekin dator, eta Mariok distantzia gehiago behar du oztopo hauek gainditzeko.

Figure 13. Marioren erorketa atzerapen handiarekin

Demagun Mariok ezkerretik salto luze bat egin duela, eta behera erori dela. Ekintza distantzia gutxi gora-behera 160 pixel dira, Mario ausazko ekintzak probatzen hasten den distantzia. Ekintza distantzia hau motzegia da goran ikusten den kasuan, Mariok salto egin

duenetik erori den arte 200 pixel badaude, berdin du zer egiten duen ekintza distantzian. Egin behar ez zuen jauzia segurtasun distantzian dago, eta Mariok ekintza hori egitea segurua dela uste du. Honen ondorioz aukeraturiko ekintza distantzia ezin daiteke txikiegia izan, horrela Mariok gaizki egin duen ekintza detektatzeko aukera du.

Honek aldi berean beste arazo bat sortzen du: ekintza distantzia handiegia bada Mariok jadanik pasatako oztopoak berriro gainditu behar ditu ausazko ekintzak eginez. Hainbat oztopo oso gertu dauden kasuetan asko geldotzen du ikasketa honek.

Figure 14. Oztopo ugari segidan

Arazo honi aurre egiteko posiblea da ekintza distantzia hau dinamikoa izatea. Hasiera batean txikia izan daiteke, eta puntu batetik igarotzea asko kostatzen bazaio pixkanaka ekintza egoera handitzen joango litzateke. Oztopoa igaro ondoren berriz txikituko litzateke ekintza distantzia.

Maila bereziak:

Hiru maila gainditzea oso zaila da une honetan eta beste mailek baina denbora gehiago behar dute (4-4, 7-4 eta 8-4), hauek ezaugarri interesgarri bat dutelako. Maila hauek labirinto moduko puzzle bat dute, hainbat bide dituzte eta Mariok ez badu bide zuzena (edo bide sekuentzia zuzena) hartzen puzzlearen hasierara bueltatzen da. Maila hauetan Markov propietatea galtzen da, aurrera jarraitzeko bidea iraganean hartu duen bidearen menpe dagoelako.

Gainera, espazioa ez da euklidearra. Mario etengabeko ziklo batean harrapatuta geratu daiteke, maila amaierara iritsi ezinik. Espazioaren forma zilindro baten modukoa da zati batzuetan, eta puzzlearen soluzioa aurkitu eta gero aterako da ziklotik.

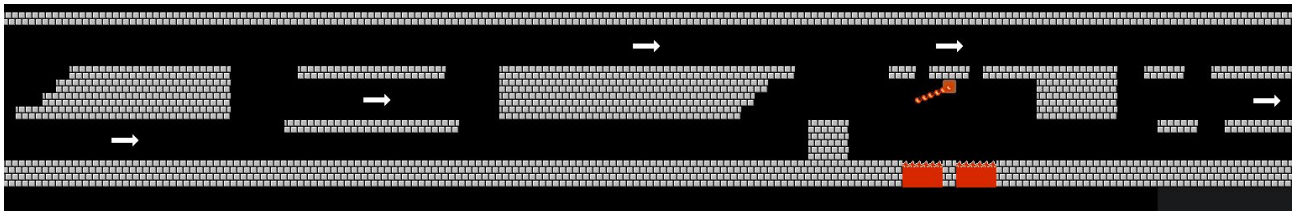


Figure 15. maila 7-4, bide zuzena erakutsiz

Teorian maila hauek ez dira ezinezkoa algoritmoarentzat, baina zorte handia izan beharko luke, ausazko ekintzak eginez bide zuzena aukeratuz. Ikasketa bermatuta geratzen da bide zuzenak hartu arte. 4-4 eta 8-4 mailek beste berezitasun bat dute: aukeratu diren ekintzak aldatu behar dira, behera eta ezkerra botoiak gehituz.

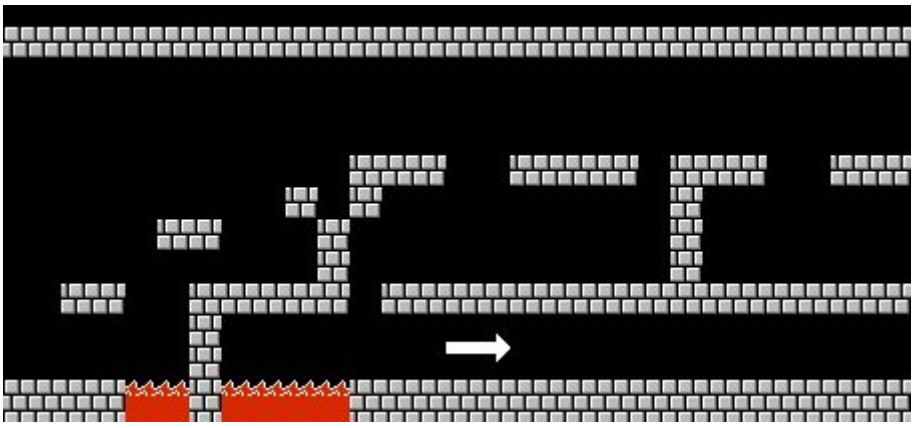


Figure 16. maila 4-4

4-4 mailan ezkerrera mugitu behar du Mariok bide zuzena aukeratzeko. Interesgarria iruditu zait datu hau, jokoa pasatzeko gutxienez ezker botoia behin zapaltzera behartuta dago Mario.

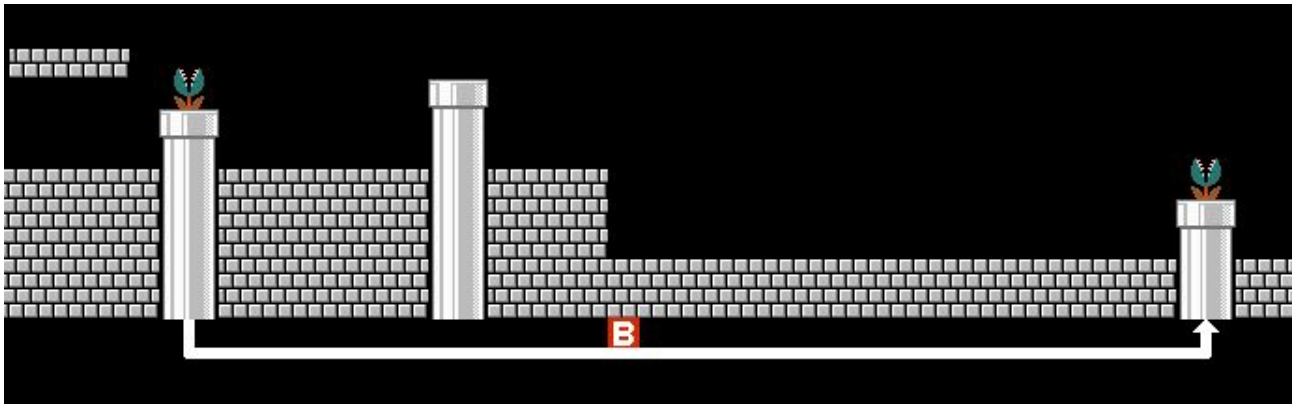


Figure 17. maila 8.4

8-4 maila gainditzeko hodiak erabili behar dira aurrera jarraitzeko. Ez bada hodia hartzen eta B puntutik igarotzen bada Mario, atzera garraiatuko du. Horrelako hiru puntu daude maila osoan, eta gainera hodi batzuek mailaren hasierara eramaten dute ere. Beraz, gutxienez “behera” botoia hiru aldiz zapaldu behar da jokoa pasatzeko.

Maila hauetan ikasteko behar duen denbora txikitzeko soluzio partzial bat aurkitu da, baina ez da oso eraginkorra. Mariok lortzen duen saria balio negatibo handia bada (-100 adibidez) labirintoaren hasierara bueltatu dela esan nahi du, eta posiblea da posizio honetan checkpoint bat jartzea, maila bertatik hasteko. Hurrengo partidetan hau gertatzen denean, partida berriro hasiko da zuzenean eta ez da ikasitako informazioa q-taulan gordeko, konbinazioa ona ez dela ikusi delako eta hurrengo partidetan ekintzak ausazkoak izatea nahi delako segurtasun distantziaren barruan. Hau eginda Mario espazio euklidear batean mugituko dela eta ez dela etengabeko ziklo batean geratuko bermatzen da eta entrenamendu denbora txikitzen da ere.

Ala ere ikasketa oso bermatuta dago labirinto barruan, ausazko ekintzak egin behar dituelako zorte ona izan arte bertatik ihes egiteko. Jasotzen duen saria denboran oso atzeratuta dator maila hauetan, eta honek muga handi bat suposatu du ikasketa abiaduran.

2.12 Hobekuntza posibleak

Q-taula eta algoritmoa Lua lengoaia erabilita inplementatu da. Programa hau Python bidez inplementatzen bada (zerbitzarian, eta ez bezeroan) posiblea izango litzateke hainbat makina erabiliz ikasketa paralelizatzea, denek ikasketa taula bera erabiliz.

2.13 Emaitzak

Q-learning algoritmo honekin lortutako emaitzak esperotakoa baina hobeak izan dira. Jokoak 32 maila ditu guztira, eta 29 gainditzea lortu du. Egoera bezala frame bakarra erabili da, eta ez lau frame. Lau frame erabilita Markov propietatea indartsuagoa da, baina exekuzioa geldotzen zuen, 80-90fpstik 40-50ra jaitsiz. Honen ondorioz denbora bikoitza edo hirukoitza behar zuen maila bat pasatzeko, teorian hobea den egoera definizioa erabiliz. Determinismo eta segurtasun distantziari esker ez da beharrezkoa izan Markov propietatea zorrotza izatea.

Denbora aldetik oso azkar ikasten du mailak pasatzen. Maila errazak pasatzeko 2 minutu inguru behar ditu. Zailenak pasatzeko berriz, hamar minutu ondoren pasatzea lortu ditu, baina zorte txarra izanez gero hau luzatu daiteke. Maila bat lehen aldiz pasatzen ikasten duenean, ez du gehiago huts egingo maila horretan, baina ez du ikasten jarraituko soluzio hori optimizatzeko asmoz, ikasketaren mugara iritsiz.

Bideo [9] honetan ikus daiteke nola pasa dituen 29/32 maila.

[9] https://drive.google.com/file/d/1qhg9tA_Yc1n6WxYn3tq_NRqexmkUN7Hv/view?usp=sharing

3. Sare Neuronala

Sarrera

Q-learning algoritmoak muga bat dauka: egoera berrietan ez daki zein ekintza izan daitekeen hoberena, ez dakielako nola kalkulatu edo estimatu ondoren pasako dena. Overfitting totala egiten du, eta modu bakarra ikasten du maila pasatzeko, honek muga handia suposatzen du. Mario jokoaren kasuan ondo funtzionatzen du honek, baina egoera espazio konplexua duten inguruneetan ezinezkoa da horrelako soluzio bat aplikatzea. Kamara bat erabiliz ate bat ireki nahi duen robotak zailtasunak izango lituzke adibidez.

Beraz, hurrengo urratsa sare neuronal batekin konbinatzea izango litzateke. Honek abantaila batzuk ditu: Q-taula ez da erabili beharko, erabilitako memoria gutxituz. Garrantzitsua hurrengoa izango da: Bellman ekuazioak lortzen duen balioaren hurbilketa lortzea. Egoera berrietan zein izan daitekeen ekintza hoberena asmatzen saiatuko da, etorkizuneko sari metatua estimatuz. Hau lortzeko modelo bat sortzen da, non input moduan pixel balioak pasatzen diren eta output moduan ekintzen balioak itzultzen dituen. Mario ikasten doan eñean, modeloaren hurbilpena optimizatzen joango da.

Konputazionalki garestiagoa izatea espero da, pauso bakoitzean kalkulu gehiago egin behar direlako, baina orokorrean funtzionamendua hobea izango dela suposatzen da.

Neurona sare batek hiru osagai nagusi ditu:

Sarrera geruza: Egoeraren datuak input moduan sartzen da. x1, x2 … x_n pixel erabiltzen dira.

Ezkutuko geruza: gutxienez geruza bat egongo da, baina normalean geruza bat baino gehiago egotea da ohikoena (sare hauek “deep” izena dute). Geruza bakoitzak ezaugarri desberdinak izan ditzake, hainbat mota daudenez.

Irteerako geruza: y1, y2, … yn ekintza espazioak osatzen du. Adibidez, y1 = A zapaldu izan daiteke, y2 = B + eskuin zapaldu etab. Aurreko diseinua erabiliz gero, lau aldagai izango ditu irteerak.

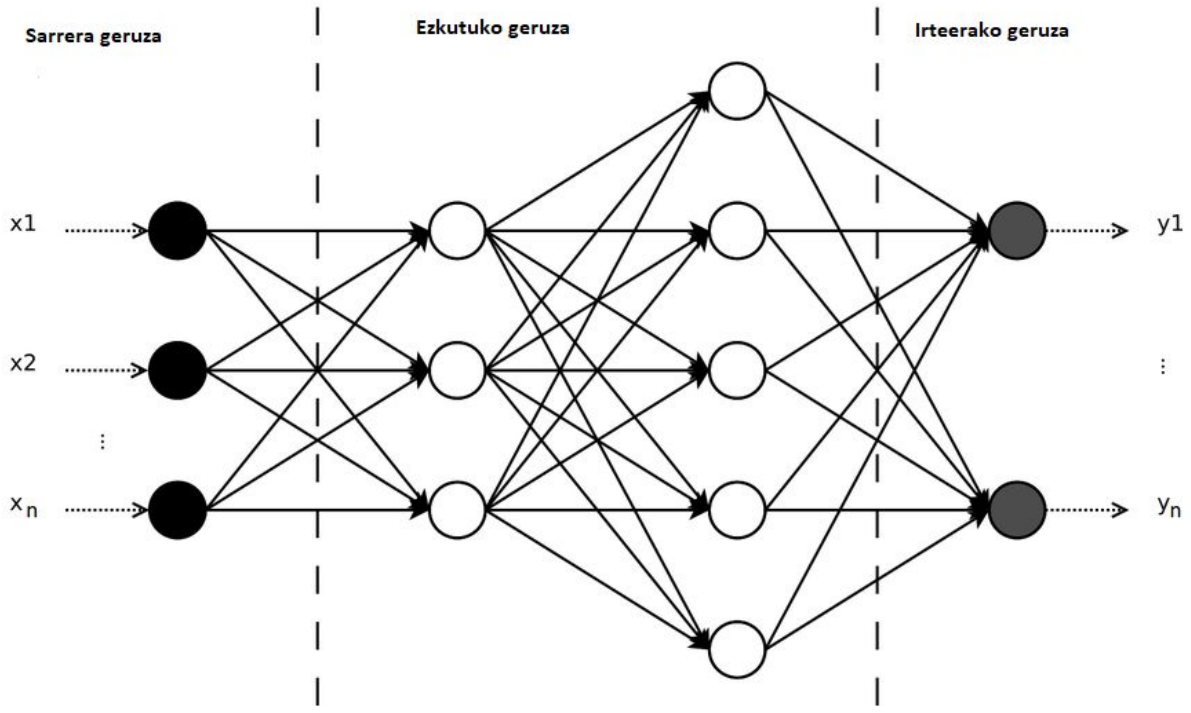


Figure 18. Sare neuronal baten egitura

Neurona bakoitzak balio bat du eta hurrengo geruzako neuronekin konektatuta dago, konexio hauek pisu bat dutelarik. Sarrera prozesatu egiten da eta irteerako geruzan ekintza bakoitzak zenbaki bat jasotzen du. Ekintza guztien artean zenbakirik altuena duena erabiliko da ekintza hoberena aukeratzeko (greedy hurbilketa).

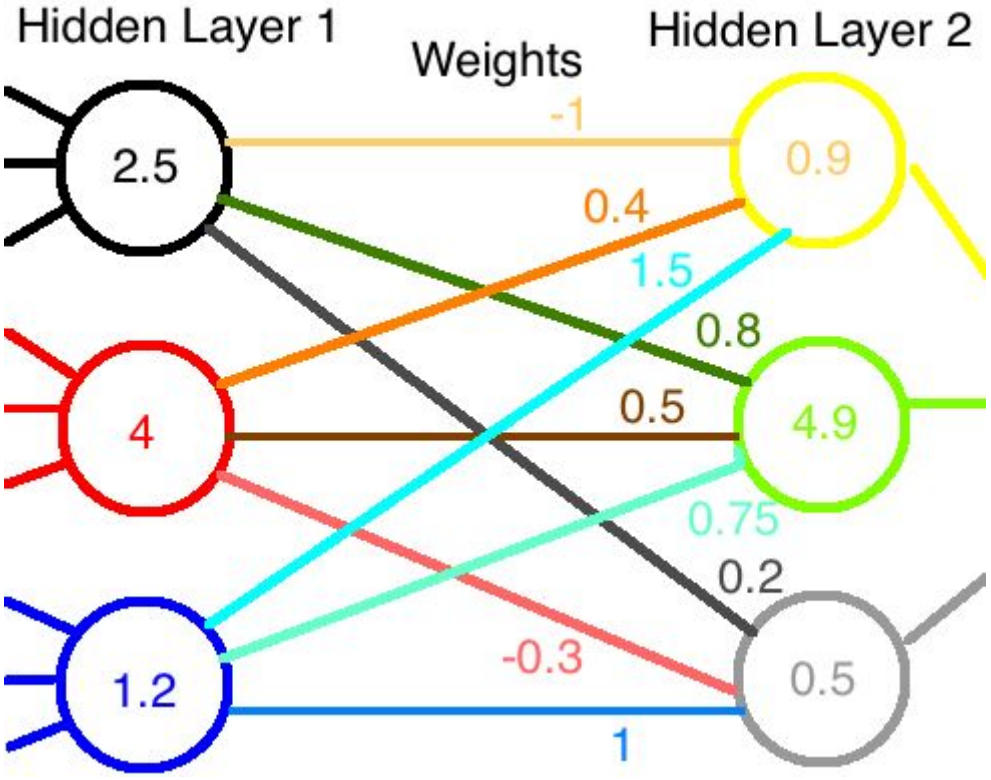


Figure 19. Neuronen balioak prozesatzen

Irudian ikusten den moduan, sarrerako balioak pisu batzuekin biderkatzen dira, eta azkenik denak batzen dira. **Tentsoreak** matrize antzeko elementuak dira, baina GPU-a erabiliz konputazio paraleloa erabiltzeko diseinaturik daude. Aurreko sarea tentsore hauek erabiliz irudikatu daiteke:

$$\begin{bmatrix} -1 & 0.4 & 1.5 \\ 0.8 & 0.5 & 0.75 \\ 0.2 & -0.3 & 1 \end{bmatrix} \begin{bmatrix} 2.5 \\ 4 \\ 1.2 \end{bmatrix} = \begin{bmatrix} 0.9 \\ 4.9 \\ 0.5 \end{bmatrix}$$

Figure 20. Tentsore bidezko errepresentazioa

Errefortzu bidezko ikasketa **ez da egonkorra,** pauso bakoitzean egoera berriak azaltzen direlako. Gainbegiratutako ikasketa eta gainbegiratu gabeko ikasketan orokorrean datu base finko bat erabiltzen da, eta horrela errore funtzioaren konbergentzia bermatuta dago, baina kasu honetan aldaketa batzuk egin behar dira ikasketa hau egonkorra izateko eta kontrolpean edukitzeko optimizazio pausoa.

3.1 Markov propietatea

Q-learning erabiltzerakoan Markov propietateak garrantzia izan du, baina ez hainbeste. Sare neuronalean berriz funtsezkoa da Markov propietatea zorrotza izatea egoera trantsizioetan, bestela programa nahasteko aukera dago.

Horretarako lau frame erabili dira egoera bakoitzeko, algoritmoak noranzkoak eta abiadurak detektatzeko. Azken ekintza salto izan den ala ez kodifikatu da ere irudian, jauziaren kontrolarekin arazorik ez izateko.

Pantailatik kanpo kargatzen den informazioa konpontzeko soluzio efizienterik ez da aurkitu, honen ondorioz kasu batzuetan aztertuko diren egoera kopurua handiagoa izango da, eta ikasketa geldotuko du hobeto ikasi arte arazo horri nola aurre egin.

3.2 Egoera

256x224 tamainako lau irudi erabiltzea egoera bakoitzeko gehiegi izan da. Egoeren trantsizioak GPU-an gordetzen direnez honek esperientzia memoriaren tamaina mugatzen du. Arazo honi aurre egiteko irudiaren eskala aldatu da, 84x84 tamainara jaitsiz eta gris eskala erabiliz. Honi esker memoria handiagoa erabili ahal izango da, eta sareak egin behar dituen eragiketa kopurua murrizten da ere.

Tamaina jaisterakoan zehaztasuna galtzen denez, lehenik irudiaren mozketa egin da informazio garrantzitsuena erabiltzeko. Irudiaren gaineko lerroak ez dira beharrezkoak, bertan txanpon kopurua eta bestelako informazio datuak azaltzen dira. Mario ezkerrera mugitzen ez denez pantailaren erdialdean egongo da kokatua, horregatik ezkerretako hainbat zutabe ez dira garrantzitsuak izango.

Figure 21. Pixel garrantzitsuenak erabiltzen

3.3 Saria

Hasierako inplementazioan saria eta Marioren posizioa gauza bera izateak abantailak ekarri ditu sistema efizienteago bat inplementatzean, baina sistema horiek ezin direnez sarearekin erabili sariaren definizioari aldaketa batzuk gehitu zaizkio.

Mario hiltzen denean penalizazio bat jasoko du eta saria -10 izango da. Honekin programak egoera arriskutsuak hobeto detektatzea espero da.

Mario jokoan segundo bat pasatzen da 24 frame pasatzen direnean. Denbora hori penalizazio moduan erabiltzen bada, 24-ren multiploak diren frame guztiek penalizazio bat sufrituko dute, eta honek ez du zentzu handirik. Alternatiba bat aurkitu da pasatako denborari garrantzi gehiago emateko: Marioren abiadura 0 bada (edo ez bada mugitu) penalizazio txiki bat jasoko du -1 moduan.

Aldaketa hauekin saria [-10, 15) tartean egongo da. Tarte hau handia ez izatea gomendatzen da, ikasketa egonkorragoa izan dadin. Demagun -1000 penalizazioa erabiltzen dela Mario hiltzean, eta sareak 5-eko saria lortzea espero zuela. Honek sortzen duen errore absolutua 1005-koa da, eta errore koadratiko funtzioa erabiltzen bada gradienteak pisu matrizea asko aldatuko du, errore handiagoak sortuz beste egoeretan eta posiblea da errore funtzioaren konbergentzia ez aurkitzea.

3.4 Politika

Sareak ε-greedy erabiltzen du une guztietan, eta honek ikasketa mantsotu dezake. Epsilon balioa geroz eta txikiagoa izango da, 5% izatera heldu arte, horrela egoera berriak begiratzen jarraituko du, eta gehiago ikasten. Kasu batzuetan garrantzitsua da ekintza kate zehatz bat exekutatzea, edo ekintza bat ez egitea beste kasu batzuetan, eta zorte txarra izanez ausazko ekintza egiten bada Mario hil egingo da.

Irudian ikusten den moduan, Mariok salto luzea egin behar du. Ausazko ekintza egiten badu 50% probabilitatea dauka salto botoia askatzeko eta zulotik behera erortzeko. Beste kasu batzuetan Mariok bere gainean arrisku bat du, eta salto egiten badu unean hilko da.

Ikasketa egonkorragoa izateko aldaketa bat egin da ausazko ekintzaren aukeraketan, ekintzak ekiprobableak izan ordez bakoitzak probabilitate pisu bat izango du, honen esperotako q-balioaren arabera.

Demagun ekintza bakoitzaren q-balioak hauek direla: **array = (-20, -10, 300, 250)**. Zentzua du 0 edo 1 ekintza ez aukeratzea ausaz, ez delako espero asko ikasiko duenik bide hori jarraituz. Zenbaki hauen artekoen minimoa **x** zenbaki negatibo bat bada, **array - 2x** kalkulatu behar da lehenik, alde positibora eramateko datuak. Aurreko kasuan txikiena -20 da, beraz **array + 40 = (20, 30, 340, 290)** balioak lortuko dira. Ondoren, array hau elementu guztien baturarekin zatitzen da, elementu guztien batura 1 izateko, **array / 680 = (0.03, 0.04, 0.50, 0.43).** Hauek izango dira ausazko ekintzak aukeratzeko erabiliko diren probabilitateak. Aldaketa hau eginez espero da Mario ez hiltzea hainbeste ondo ikasita dituen arriskuen aurrean.

Orokorrean, ε-greedy politikak arazoak ekarri ditu eta beharrezkoa izan da aldaketak aplikatzea eraginkortasuna hobetzeko.

3.5 Ereduaren egitura

Eredua sortzerako garaian badirudi ez dagoela oso argi zein parametro erabili behar diren. Eredu batzuk beste batzuk baino hobeak dira, eta oraindik aurkitu ez diren ereduak egon daitezke ere. Denbora asko behar denez eredu bat sortzen eta optimizatzen, proiektutik at geratu da hau eta Atari bideo-jokoetan jokatzen ikasten duen eredu bat erabili da, antzeko ingurunean aplikatu delako (irudi bidezko kontrola bideo-jokoetan).

Honek sarrera moduan 84x84 tamainako lau irudi hartzen ditu. Ondoren, hiru konboluzio geruza ditu. Konboluzioaren tamaina geroz eta txikiagoa da (8x8, 4x4, 3x3) eta datu kopurua murrizteko stride erabiltzen da (batzuetan honen ordez maxpooling ere erabili daiteke).

Bi guztiz konektaturiko geruza daude amaieran. Lehendabizikoak 512 aldagai ditu, eta azkenak 4 aldagai, Marioren ekintza bakoitzerako aldagai bat. Beraz, aurreraka pase bat eginda ekintza guztien q-balioak kalkulatzen dira. Azken geruzan izan ezik, ReLU aktibazio funtzioa erabiltzen da, funtzioa ez izateko lineala.

Geruza hauen azalpen sakonagoa Eranskinak atalean dago.

Modelo hau erabiliz, hainbat Atari jokoetan jokatzen ikasi zuen algoritmoak, zailtasun handienak urruneko estrategiak behar dituzten jokoetan aurkituz.

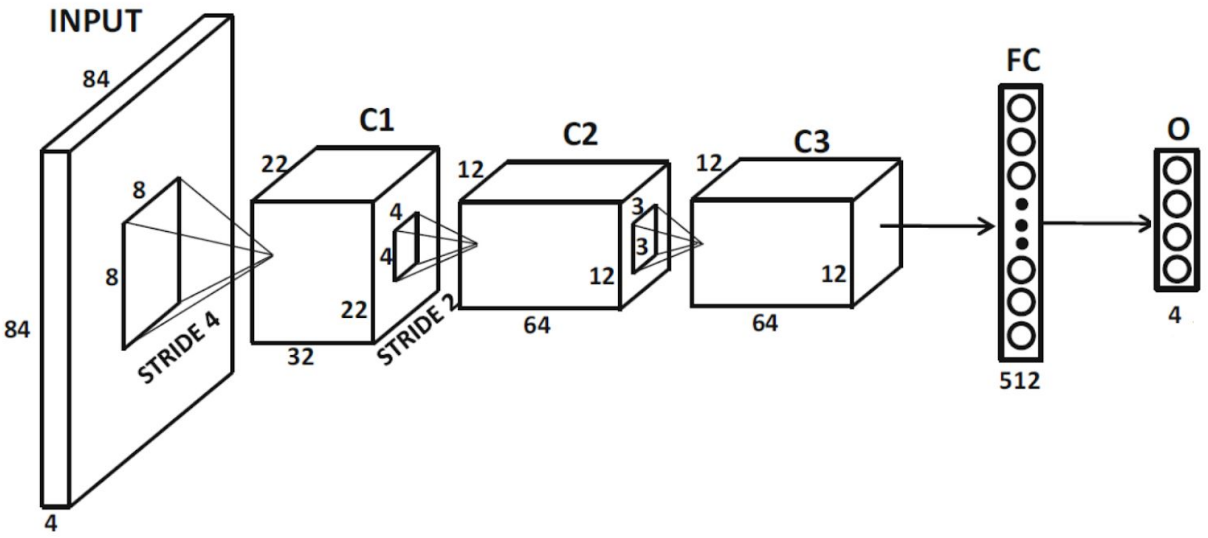


Figure 22. Sarearen arkitektura.

Beste sare arkitektura batzuk ere begiratu dira, baina konboluzio sare gehienak irudiak sailkatzeko erabili dira, eta ez errefortzu bidezko ikasketan. AlexNet arkitektura adibidez irudiak sailkatzeko erabili zen, bost konboluzio geruza ditu hasieran, eta hiru guztiz konektaturiko geruza amaieran. Geruza bakoitzaren amaieran ReLU aktibazio funtzioa erabiltzen da, stride eta MaxPooling (MP) erabiltzen ditu datuen dimentsioak jaisteko.

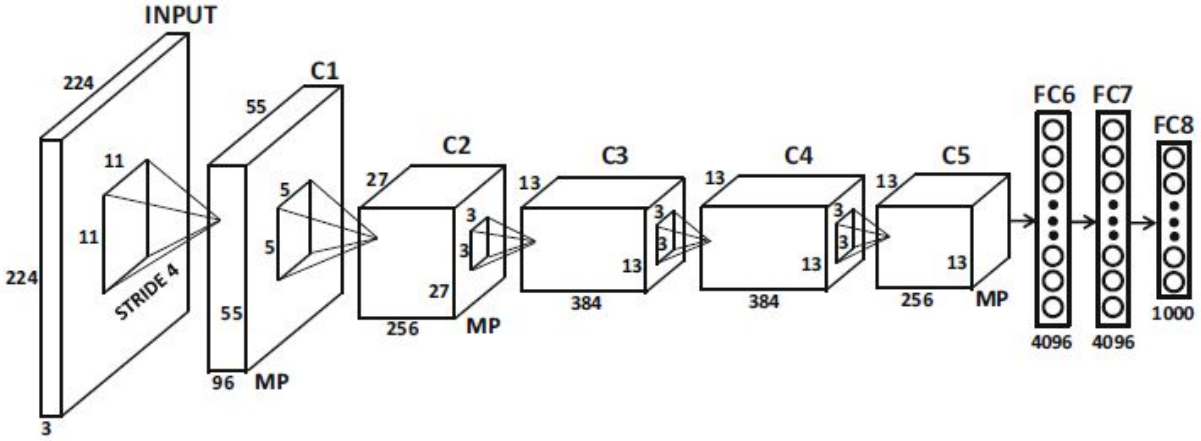


Figure 23. AlexNet arkitektura

LeNet sarea berriz, eskuz idatzitako idazkiak detektatzeko erabili zen.

Beste sare batzuek milaka konboluzio geruza izan ditzakete, datuen normalizazioak erabiliz “vanishing gradient” arazoa ekiditen duena. ResNet sareak 152 geruza ditu adibidez, “skip connection” izeneko konexioak erabiltzen dira gradiente arazoak ekiditeko, non geruza batzuk hainbat geruza desberdinetara konektaturik dauden. Beste batzuek geruzetako balioak normalizatzen dituzte arazo horri aurre egiteko.

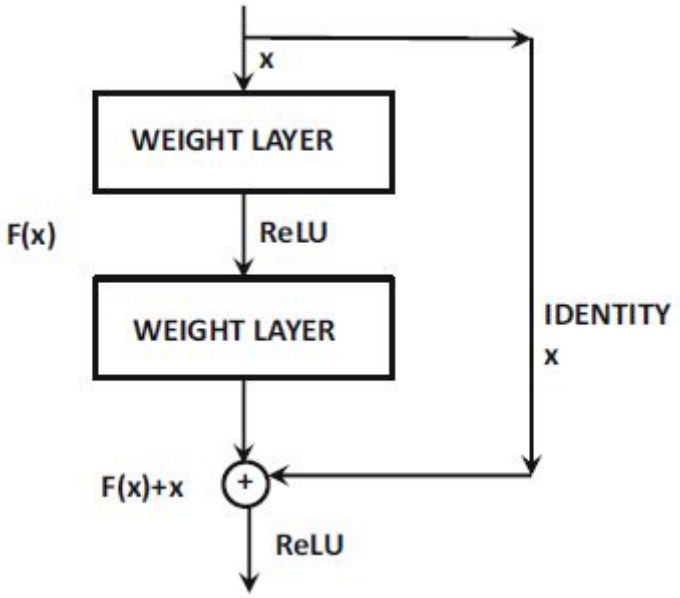


Figure 24. Skip connection

3.6 Backpropagation

Sare neuronalak aurreraka pasea egitean duenean konputazio grafo bat sortzen da (Eranskinak atalean dago konputazio grafo honen irudia). Ondoren, atzeraka pase bat egiten da deribatu partzialak kalkulatuz eta katearen erregela aplikatuz. Honi esker pisu aldagai bakoitza handitu ala txikitu egin behar den jakin dezakegu, errore funtzioa txikitzeko. Hau egiteko beharrezkoa da funtzioak deribagarriak izatea puntu guztietan. Pytorch erabiltzerako garaian pisu tentsoreek “requires_grad = True” aldagaia izango dute, honek tentsore horren gradientea kalkulatu behar dela irudikatzen du. Sarrerako datuen gradientea ez da kalkulatu behar adibidez.

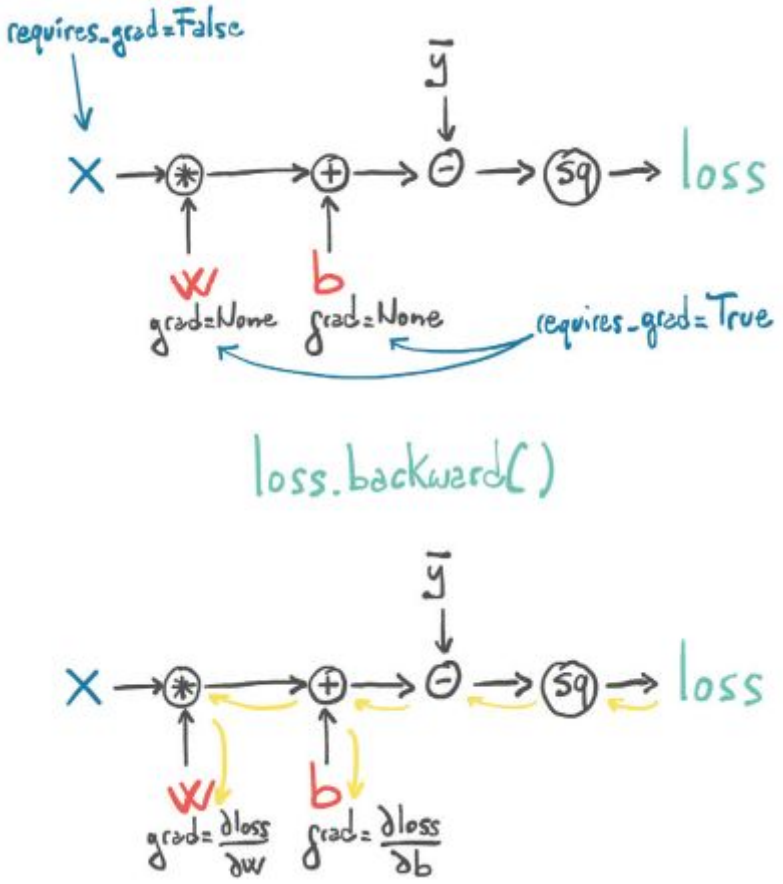


Figure 25. Backpropagation konputazio grafoan zehar

Gradiente hauek kalkulatu ondoren erraz jakin dezakegu pisu aldagai bakoitza handitu ala txikitu egin behar den errore funtzioa minimizatzeko. Hau egiteko optimizagailu bat erabiliko da.

3.7 Errore funtzioa

Neurona sarearen zehaztasuna kalkulatzeko eta optimizazioak egiteko errore funtzioa definitzea beharrezkoa da. Horretarako Bellmanen ekuazioa erabiliko da:

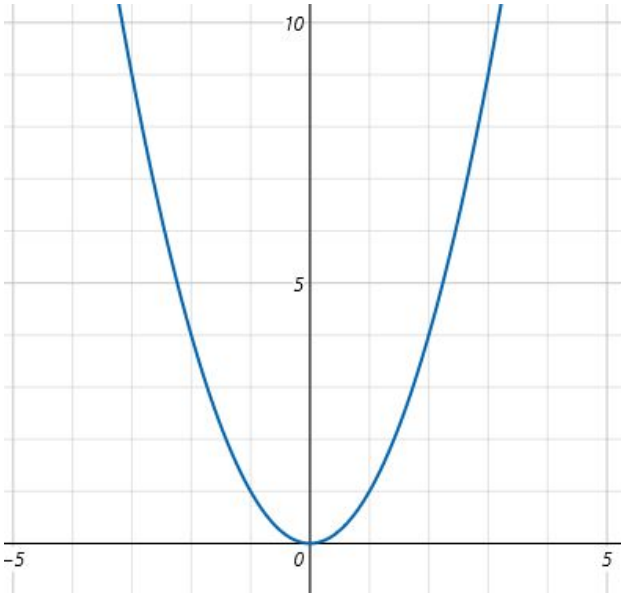
$$Q^{new}(s_t, a_t) \leftarrow \quad r_t \quad + \max_a Q(s_{t+1}, a)$$

Ikusten den moduan, bi ezezagun daude ekuazioan: $Q^{new}(s_t,a_t)$, uneko egoeraren balioa izango dena, eta $\max_a Q(s_{t+1},a)$, hurrengo egoeraren balioa. Sarea balio hauek zeintzuk diren asmatzen saiatuko da, eta errorea kalkulatzeko bi balio hauen arteko diferentzia kalkulatuko da. Beraz, errorea = $Q^{new}(s_t,a_t)$ - (r_t + $\max_a Q(s_{t+1},a)$) izango da.

Batez besteko errore koadratiko funtzioa erabiltzea aukera bat da:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

Funtzio hau erabilita errorea beti positiboa izango da, eta errore handiek balore handiago bat sortuko dute. Honek arazoak sortu ditu konbergentzia aurkitzeko garaian, errore altuek sarearen pisu matrizean eragin handiagoa dutelako, eta horren ondorioz beste egoerak sortutako errorea handitzen duelako.



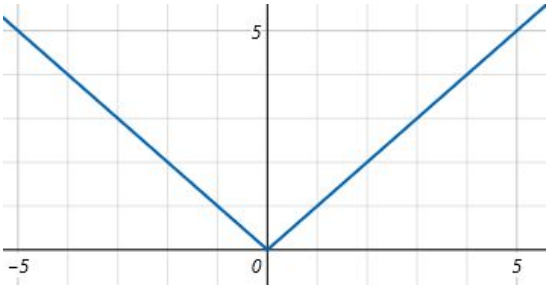
$f(x) = x^2$ funtzioaren grafika. Honen deribatua $f'(x) = 2x$ da, beraz errorea geroz eta altuagoa izan, malda handiagoa izango da eta baite ere backpropagation pausuan kalkulatuko den gradientea. Honen arriskua “exploding gradient” da, non jauziak geroz eta handiagoak diren, eta minimoa aurkitu beharrean dibergentziara doa.

Honen soluzio posible bat errore funtzioak atzeraka egitean sortzen duen gradientearen baloreak tarte txiki batera mugatzea izan daiteke, aldaketak leunagoak izateko.

Exploding gradient arazoa ekiditeko beste errore funtzio bat erabili da: Batez besteko errore absolutua.

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

Funtzio horrekin errore handiek eta txikiek magnitude bereko aldaketak sortuko dituzte sarearen pisuetan.



f(x) = |x| funtzioa. Honen deribatua f'(x) = 1 baldin eta x > 0, eta -1 baldin eta x < 0 izango da. Errorea 10 bada edo 10000 bada hauen deribatuak berdinak izango dira, eta sarearen pisuetan sortzen duten aldaketak berdinak izango dira. Honek ikasketa geldotu dezake errore handiak sortzen dituzten egoerei ez zaiolako beste egoerei baina garrantzi handiagoa ematen, baina lehentasun bidezko esperientzia memoria erabiliz arazo honi aurre egingo zaio.

Funtzio honek ere arazo bat dauka: ez da deribagarria 0 puntuan. Arazo honi aurre egiteko soluzio misto bat erabili da, **Huber loss** izenekoa: [-1, 1] tartean errore koadratikoa erabiliko da, eta bestela errore absolutua. Funtzio hau deribagarria da puntu guztietan, eta ez da hain sentikorra izango muturreko datuekin.

3.8 Optimizazio pausoa

Backpropagation eta katearen erregelari esker pisu aldagai bakoitza handitu ala txikitu egin behar den jakin dezakegu. Aldaketa hauek aplikatzeko optimizagailu bat erabiltzen da, oinarrizkoena **gradient descent** izenaz ezagutzen da, maldan bera joaten dena minimoa aurkituz.

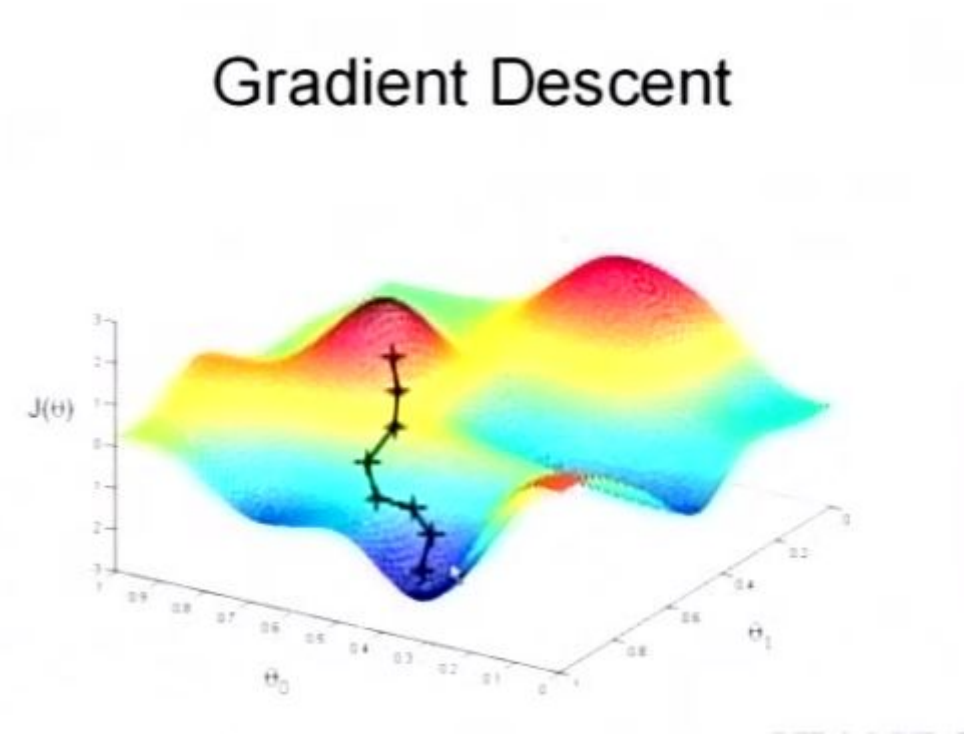


Figure 26. Gradient Descent erabilera errorea minimizatzeko (bi aldagaiekin)

Gradient descent algoritmoak muga batzuk ditu eta kasu batzuetan ez du minimo totala aurkitzen. Inflexio puntuetan deribatuaren balioa zero denez baliteke ez aurkitzea maldan behera jarraitzeko bidea. Beste kasu batzuetan minimo lokal bat aurkitzen du, eta ezingo du minimo globala aurkitu.

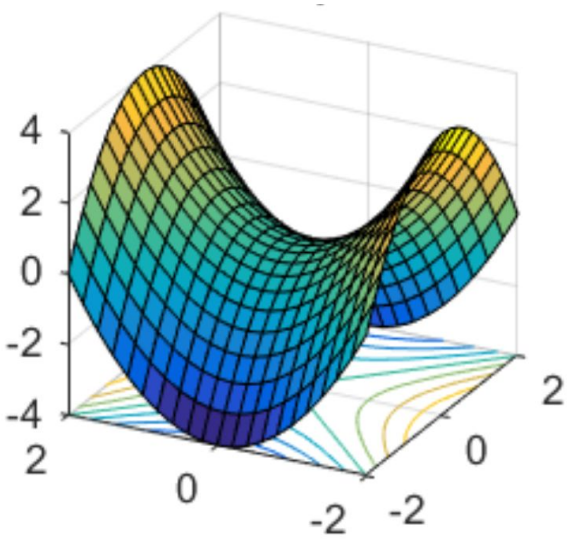


Figure 27. Inflexio puntua

Arazo hauek ekiditeko hainbat algoritmo daude, horietako batek (**momentum** izena duenak) jaitsieraren abiadura kontuan izatea, kanika bat erortzen egongo balitz bezala. Beherako puntura iristean, abiadura hori galdu beharrean maldan gora joaten saiatzen da, horrela minimo lokaletatik ateratzeko aukera edukiz.

Batzuetan aldagai batzuk gutxietan eguneratzen dira, hau konpontzeko aldagai bakoitzarentzat ikasketa ratio desberdin bat definitu daiteke, baina hau ez da optimoa. Askotan eguneratzen diren balioak etorkizunean gutxiagotan eguneratu behar dira, gutxi eguneratzen diren aldagaiak ere eguneratzeko. Horretarako **Adagrad** algoritmoak aurreko gradienteen batura erabiltzen du bide berriak aurkitzeko. Honek arazo bat du, batzuetan gradiente hauek oso handiak direlako programak maneiatzeko eta funtzionatzeari uzten dio. **RMSProp** algoritmoak arazo honen aurrean “decay rate” erabiltzen du, aspaldiko gradienteak pixkana ahaztuz eta garrantzi gehiago emanez duela gutxiko gradienteei.

Azkenik, **Adam** algoritmoak aurreko teknika guztien hoberena hartzen du. Alde batetik abiadura kontuan hartzen du azkarrago jaisteko, eta hainbat norabidetara joateko ahalmena du ere RSMProp bezala. Honi esker oso optimizagailu indartsua da, eta hau da programak erabiliko duena.

3.9 Esperientzia memoria

Trantsizioak (s_t, a_t, s_{t+1}, r_t) informazioa buffer edo lista batean gordetzen da, eta buffer honetako sorta bat erabiltzen da sarea entrenatzeko pauso bakoitzean. Honek hainbat abantaila ditu: azken partidako egoeren korrelazioa jaisten da entrenatzerako garaian (algoritmoak hau jasotzea espero du), ikasketa azkartzen du sorta txikiak erabiliz, eta iraganeko informazioa erabiltzen du, ez ahazteko garrantzitsua den informazioa. Sortaren tamaina aukeratzeko garaian gauza pare bat kontuan hartu behar dira. Tamaina oso txikia bada, pauso bakoitzean ez du asko ikasiko. Bestalde, tamaina handia bada (memoria osoa erabiltzea da muturreko adibidea) errore funtzioa azkar optimizatuko luke gradientearen kalitatea hobea delako. Teorian errore funtzioa minimizatzea ona dela dirudi, baina horrek overfitting sortuko luke, lortu nahi den orokortasuna galduz.

Replay memoriak arazo bat dauka: ausazko sorta bat hartzen da entrenamendu pausu bakoitzean baina egoera guztiak ez dute garrantzi bera. Amaierako egoerak adibidez, askotan ez dira sortaren barruan egongo, eta hauek informazio baliagarria izan dezakete ikasteko.

Soluzio moduan lehentasun sistema bat erabiliko da. Entrenamenduan errore handien sortzen duten egoerei lehentasuna emango zaie, gehiagotan agertzeko eta hauei buruz ikasteko. Sortako elementu guztiek probabilitate bat izango dute sortan sartzeko, ez da egongo %0 izango duen elementurik.

Egoera berriak memorian txertatzean lehentasun maximoa emango zaio, hurrengo iterazioko sortan azaltzeko probabilitateak asko handituz.

Optimizazio aukera oso ona aurkitu da hemen. Entrenatzeko sorta bat erabiltzerakoan, memoriatik atzitzen ziren datuak, ondoren tentsore bat sortzeko eta GPU-ra pasatzeko datuak. GPU-aren erabilera oso txikia zela nabaritu zen (1% inguru) eta sortaren tamaina handituz programa asko geldotzen zen, txartelari lana eman gabe.

Honen arrazoia sortaren datuak etengabe GPU-ra pasatzen ari direla da, eta transmisio abiadura hori nabaritzen zen. Datuak behin eta berriro berrerabiltzen direnez, replay memorian hobekuntza bat egin da: Memorian informazioa gordetzean tentsorea zuzenean txartel grafikora pasatzen da. Honi esker 512 tamainako sorta arazorik gabe erabiltzea lortu da eta txartel grafikoaren errendimendua handitzea. Aldaketa hauek egin aurretik, 16 sortako elementuak erabili behar ziren programaren exekuzio abiadura normal bat izateko.

Entrenatzean GPU-a memoria gabe ez geratzeko erabiliko den espazioa hasieraketan alokatuko da. Nire kasuan 6GB RAM edukita, 15.000 tamainako array bat sortu dezake gehienez 4x84x84 tamainako egoera espazioa erabiliz. Memoria betetzen denean elementu zaharrena ordezkatuko da, eta honek memoria galera arazoak ekar ditzake, azken ordu erdian erabilitako informazioa erabiltzen duelako entrenatzeko. Honek inpaktua izango du hainbat maila desberdinetan entrenatu nahi bada, pixkanaka aurreko mailen informazioa galduko duelako.

3.10 Bi sare neuronal

Sarea etengabe gaurkotzen ari denez ikasketa desegonkorra izaten da. Horretarako bi sare neuronal erabili daitezke. Sare batek, **target_network** izenekoak, parametroak izozturik izango ditu ikasketa egonkorragoa izateko. Honek etorkizunean lortuko duen sari metatua kalkulatuko du. Beste sarea, **prediction_network**, egikaritu egingo da entrenamendu pausoetan. Sare horrek uneko egoeraren sari metatua kalkulatzen du. Bi balio hauek erabiliko dira errorea kalkulatzeko. Noizbehinka target_network sarearen balioak egikaritu egin behar dira beste sarea erabiliz, horretarako bi aldagai definitu dira: update_step eta tau. update_step aldagaiak zenbat pausoro egingo den eguneraketa definitzen du. Tau balioa berriz, parametro berrien balioak kalkulatzeko erabiliko da formula honen bidez:

target_network = tau * prediction_network + (1-tau) * target_network.

Bi konbinazio erabili daitezke, update_step = 1 eta tau = 0.001 denean pauso bakoitzean target_network pixkanaka beste modeloan bihurtuko da (soft update). Tau = 1 eta update_step = 300 motako konbinazioan hainbat pauso eta gero sare osoa kopiatuko da (hard update). Emaitza hoberenak sarea guztiz kopiatzen lortu dira, egonkorragoa delako ikasketa. Balio honek handia izan behar du, bestela dibergentzia arazoak agertzen dira (10.000 pausoz behin egiten da kopia).

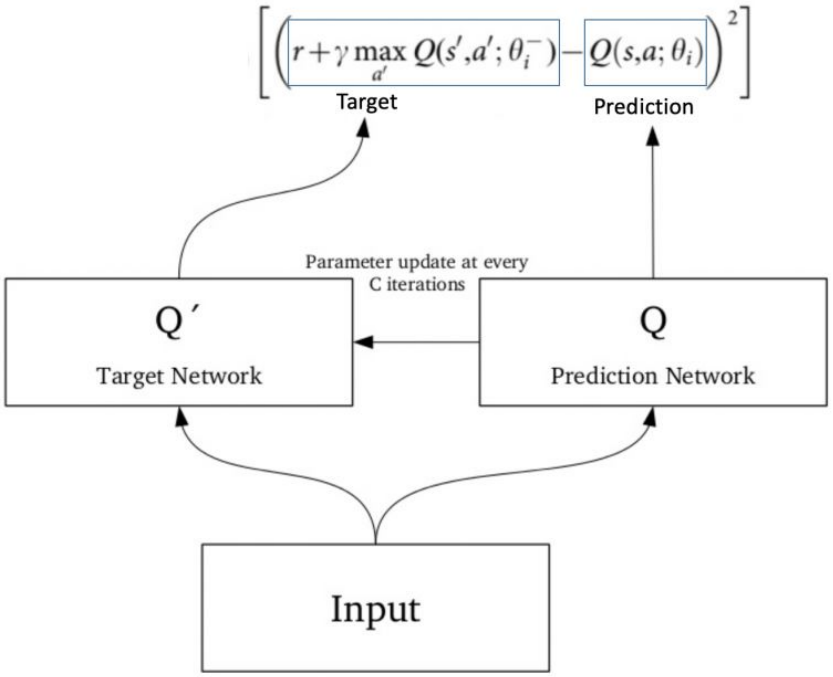


Figure 28. Bi sare neuronalen erabilera

3.11 Sare neuronalaren inplementazioa

Sare neuronala eraikitzeko algoritmo hau inplementatu da:

1. Bi sareak, esperientzia memoria eta bestelako aldagaiak hasieratu
2. Begizta nagusian sartu
3. Uneko egoera s_t gorde
4. ε-greedy erabiliz a_t ausazko ekintza edo ekintza hoberena aukeratu
5. a_t exekutatu emuladorean eta lortu den r_t saria eta s_{t+1} hurrengo egoera lortu
6. (s_t, a_t, r_t, $s_{t+1,}$ amaiera) trantsizioa esperientzia memorian gorde
7. Memoriak 32 elementu baditu, ausazko 32 (s_t, a_t, r_t, $s_{t+1,}$ amaiera) trantsizio atzitu
8. $y_j = r_j$ esleitu baldin eta s_{j+1} amaierako egoera den
   bestela, $y_j = r_j + \gamma^* \max_{a'} Q'(s_{j+1}, a')$ izango da
9. Huber(y_j - $Q(s_j, a_j)$) errorea eta honen gradientea kalkulatu backpropagation bidez
10. Adam optimizagailua erabiliz sarearen pisuak eguneratu
11. 10.000 pauso pasatzen diren bakoitzean Q' sarea eguneratu Q sarearen balioak erabiliz

Pytorch liburutegiak hainbat berezitasun ditu, kontuan hartu behar direnak inplementazio garaian.

Ekintza hoberena aukeratzeko garaian (4. pausoa) sarea.eval() erabili behar da hau erabili aurretik, honek ebaluazio modua aktibatzen du. Ekintza lortu eta gero sarea.train() erabili behar da, 8. eta 9. pausoan modu hau erabiltzeko. Honen arrazoia sarearen jokaera desberdina izan daitekeela da ebaluazio edo entrenamendu moduan, adibidez BatchNorm edo Dropout funtzioak erabiltzen badira sarean.

Ekintza hoberena aukeratzeko garaian ez da beharrezkoa sarearen gradientea kalkulatzea, ezta ere bigarren sarearen (Q') gradientea entrenamendu garaian. Horretarako torch_nograd() bloke bat sortu behar da.

```
#get_action metodoaren zati bat
#bloke hau compute_loss funtzioan ere erabili behar da, Q'-k ez
sortzeko gradientea
self.model.eval() #ebaluazio modua gaitu
with torch.no_grad():
     qvals = self.model(state.unsqueeze(0)) #gradientea ez kalkulatu
     action = np.argmax(qvals.cpu().detach().numpy())
self.model.train() #entrenamendu modua gaitu
```

Huber errore funtzioak zuzenean batezbestekoa kalkulatzen du, baina trantsizio bakoitzak sortzen duen errorea jakin behar da lehentasun bidezko memoria gaurkotzeko.

```
self.loss = nn.SmoothL1Loss(reduction = 'none') #errore funtzioaren
hasieraketa
...
loss = self.loss(curr_Q, expected_Q) #errorea kalkulatu trantsizioekin
```

reduction = 'none' erabiliz lor dezakegu hori, horrela loss aldagaia 32 elementuz osaturiko array bat izango da, trantsizio bakoitzaren errorea gordetzen duena. Ondoren, backpropagation egin aurretik batezbestekoa kalkulatu behar da.

```
loss_mean = loss.mean()#batezbestekoa kalkulatu backpropagation egiteko
```

Gradienteak kalkulatzen direnean hauek metatzen joaten dira, beraz backpropagation pausoa egin aurretik informazio hau garbitu behar da, gradientea zerora jarriz. Pytorch-ek duen berezitasun bat da, eta kontuz ibili behar da honekin, hasiera batean ez delako intuitiboa hau egin behar dela.

```
self.optimizer = torch.optim.Adam( ... ) #optimizagailuaren hasieraketa
...
self.optimizer.zero_grad() #gradienteak zerora jarri
loss_mean.backward() #gradienteak kalkulatu
self.optimizer.step() #sarearen aldagaiak eguneratu
```

3.12 Hiperparametroak

Zati honetan sareak erabiltzen dituen hiperparametroen lista aurkitzen da. Balio hauek aldatuz sarearen jokaera guztiz aldatu daiteke, eta garrantzitsua da oreka aurkitzea, konbergentzia aurkituz eta overfitting ekidinez.

observation_space = (4, 84, 84). Egoera bakoitzaren tamaina. 84x84 tamainako azken lau irudiak erabiliko dira. Irudia eskalatu behar izan da esperientzia memoria oso txikia ez izateko, baina gehiegi txikitu gabe bestela zehaztasuna galtzen da.

action_space = 4. Sarearen irteeraren tamaina, ekintza posibleen kopurua dena.

batch_size = 32. Pauso bakoitzean erabiltzen diren trantsizio kopurua entrenatzeko. Txikia bada gutxi ikasiko du, eta handia bada errore funtzioaren konbergentzia azkarragoa izango da, baina overfitting egiten du eta ez du hain ondo jokatzen ikasten.

learning_rate = 1e-4. Optimizagailuak aldagaiak eguneratzeko erabiltzen duen kopurua. Txikia bada gutxi ikasiko du, eta handia bada dibergentzia erroreak azaltzen dira.

gamma = 0.99. Bellman ekuazioan erabiltzen den balioa, urruneko balioen balorea jaisteko.

buffer_size = 15000. Esperientzia memoriaren kapazitatea, trantsizioen tamainaren eta GPU-aren RAM memoriaren araberakoa da.

epsilon. ε-greedy aplikatzean erabiltzen den probabilitatea. Hasieran 1 izango da, eta 0.99 aldiz txikiagoa izango da pauso bakoitzean, 0.05-ra iritsi arte. Behin minimora iristen denean, programak gutxiago esploratuko du.

step_update = 10000. Zenbat pauso pasata gaurkotu behar den izoztuta dagoen sarea. Balorea txikia bada ez du konbergentzia lortzeko denborarik izango, eta handia bada ikasteko denbora gehiago beharko du.

3.13 Hobekuntza posibleak

Hobekuntza posible batzuk aurkitu dira, baina zoritxarrez denbora faltagatik ez da posible izan aldaketa hauek probatzea, ordu asko behar direlako hauen eragina ikusteko.

Errefortzu bidezko DQN-ak (Deep Convolutional Network) aurrerapen gehiago izan ditu, DeepMind taldeak artikulu[10] honetan azaltzen duen moduan hainbat teknika konbinatu dira emaitza hobeak lortzeko, honek “Rainbow” izena hartu du. Artikulu horretako hainbat teknika inplementatu dira: DDQN (bi sare neuronal), lehentasunezko esperientzia memoria eta Dueling DQN (azken honekin ez dira nahiko probak egin eta kodean komentatuta dago). Beste metodoak ez dira probatu (distributional DQN, noisy DQN, A3C) baina oso posiblea da hauek hobekuntzak ekartzea programan.

[10] https://arxiv.org/pdf/1710.02298.pdf

Beste aldetik, hardwarea muga handia izan da sarearen exekuzioan, gehien bat GPU-aren memoria, honek esperientzia memoriaren tamaina maximoa mugatzen duelako. Hau konpontzeko hodei zerbitzuak erabili daitezke, adibidez AWS (Amazon), Google Cloud edo Microsoft Azure.

3.14 Emaitzak

Algoritmoak esperotakoa baina denbora gehiago behar izan du jokatzen ikasteko. Honen ondorioz ez dira probak egin maila guztietan, kasu esanguratsu batzuk bakarrik erabili dira.

Lehen maila 1-1 pasatzeko ordu bat edo bi behar ditu (lehen algoritmoak 1-2 minutu bakarrik), bi sare neuronal erabiltzeak eragina izan du hemen, 10 minutu inguru behar dituelako programak izoztuta dagoen sarea eguneratzeko. Gutxi gora-behera 500 partida behar ditu maila pasatzeko lehen aldiz (q-learningek 80 inguru), baina maila ez du egonkortasuna lortzen 1000 partida jokatu arte.

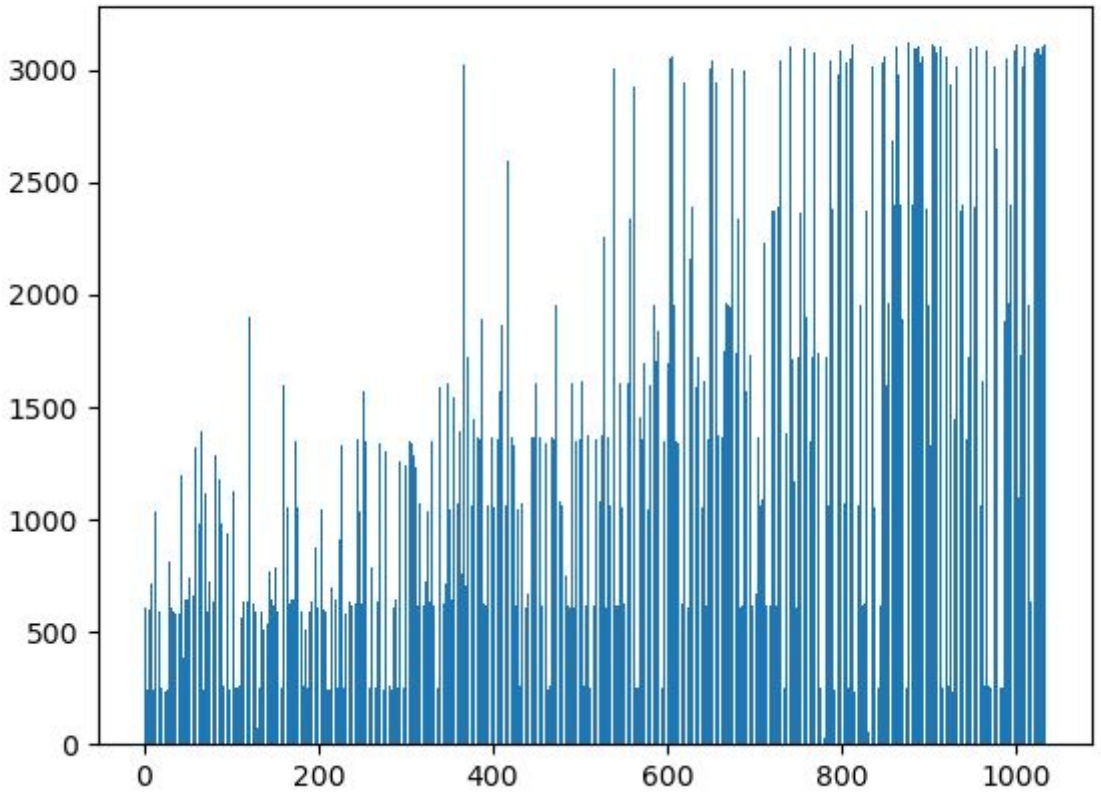


Figure 29. Batez besteko saria 1-1 mailan

Azken maila 8-3 gainditzeko arazoak aurkitu ditu programak. 12 ordu behar izan zituen maila behin pasatzeko (q-learningek 10 minutu inguru), eta gainera ez zuen lortu berriro pasatzea. Hardware mugak zerikusia izan duela susmatzen da, GPU-aren memorian 15.000 elementu bakarrik kabitzen dira, hau entrenamenduaren azken 20-30 minutu dira. Memoria muga honengatik garrantzitsua den informazioa ahazten du eta

zailtasunak ditu egoera guztien aurrean ondo erantzuteko. Mailak luzeagoak dira eta etsaiak arriskutsuagoak, eta Markov propietatea ez denez guztiz betetzen zati batzuetan zer egin behar duen ikastea asko kostatzen zaio egoeraren konplexutasunagatik.

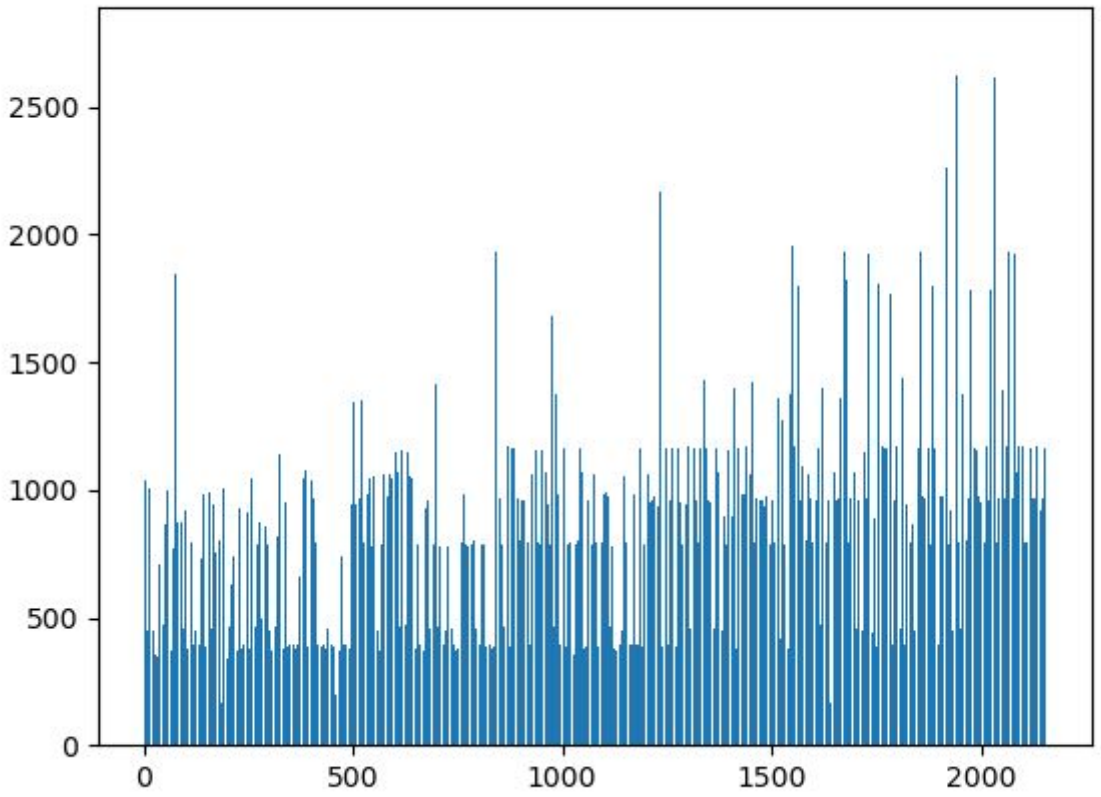


Figure 30. Batez besteko saria 8-3 mailan

Gainera, programatzerako garaian askatasun gutxiago ematen du sareak. Kaxa beltz moduko bat bezala da, eta bertan gertatutakoak sarearentzat bakarrik du zentzua. Honen portaera aldatzeko gehienbat parametroen balioek dute eragina, edo funtzioaren konbergentzia lortzeko erabili diren hainbat teknika.

Errorea kontrolpean edukitzea lortu da, batez-besteko errorea 10 baina txikiagoa izatea lortu da entrenamendu osoan zehar.

Bestalde, algoritmo honek abantaila batzuk erakutsi ditu ere. Alde batetik, lortu nahi zen orokortasuna lortu da. Posiblea da sarea eta ε-greedy erabiltzea, ausazko ekintzak egiten direnean sarea ez da galtzen eta maila gainditzeko modua aurkitzen du, egoera berrietan ere.

Q-learningek soluzio bat aurkitzen zuenean ez zuen gehiago ikasten, baina sareak ikasten jarraitzen du emaitzak optimizatuz. Honen ondorioz mailak azkarrago gainditzen ditu. Lehen maila pasatzeko q-learningek 100 segundo inguru behar ditu, eta sareak minutu batean pasatzea lortzen du. Azken mailan berriz, sareak 20 segundoko diferentzia ateratzen dio lehen inplementazioari. Jokatzen duen garaian hau nabaritzen da, Mario adimentsuagoa dela dirudi eta aukeratzen dituen ekintzak ez direla hain ausazkoak.

Azkenik, **A3** arriskua bete da, ikasten esperotakoa baina denbora gehiago behar izan dut eta ondorioz planifikatutako hainbat hobekuntza ez dira inplementatu. Hauekin sarea hobetzea espero da, baina ez da espero sekulako hobekuntzak izatea.

4. Ondorioak

Q-learning erabiliz algoritmo zahar honek overfitting egiten duela eta honen erabilpena mugatua dela ikusi da. Programa ondo dabil arazo sinple eta konkretu batzuekin, optimizazio teknikak aplikatzea posible delako ingurunearen arabera, baina egoera espazioa estokastikoa eta konplexua bada beharrezkoa da beste metodo bat erabiltzea.

Sarearen abantaila nagusia honen moldakortasuna da. Hauen desberdintasun nagusia orokortzeko ahalmena izan da, sareak eredu konplexuak sortzeko kapaza da eta honek erabilera anitz ditu hainbat eremuetan. Hau lortzeko hardwareak muga handia sortzen duela ikusi da, ordenagailu pertsonalak oraindik ez dute nahikoa potentzia ingurune konplexuetan ondo ikasteko, non beharrezkoak diren milaka GPU eta paralelizazioa erabiltzea honek eskatzen duen potentzia eta memoria arazoei aurre egiteko.

Sareak algoritmo zaharrak zituen oztopoak gainditzea espero zen, 30 urteko diferentzia dagoelako bi hauen artean, baina hau ez dela horrela izan ikusi da. Maila konplexuetan egoera kantitatea handia izan daiteke, eta zailtasunak izan ditu oztopo hau gainditzen duen eredua sortzean. Saria denboran oso atzeratuta datorren mailetan ez du arrakastarik izan ere (hiru maila bereziak), zailegia delako jakitea zein ekintzek izan duten garrantzi handien bide zuzena aurkitzerako garaian. Moldakortasun hori lortu den arren, oraindik arazo hauei aurre egiteko teknika eraginkorrik ez dagoela dirudi, eta soluzio bakarra entrenamendu denbora handitzea dela dirudi, hardware hobeagoa erabiliz eta exekuzioa azkartuz. Sarean erabili diren teknika eta optimizazioak errefortzu bidezko ikasketak dituen arazoei aurre egin beharrean errorearen konbergentzia hobetzeko erabili dira, baina oraindik ez dago soluzio egonkorrik errefortzu bidezko ikasketak sortzen dituen erronkei aurre egiteko.

Ikusmen bidezko sare neuronalak errefortzu bidezko ikasketan hobetzeko aukera handiak daudela dirudi. Une honetan beharrezkoa da hardware indartsua erabiltzea, baina muga hau gainditzen bada algoritmoak hobetuz eta behar den konputazio / ikasketa denbora behar hauek jaitsiz (edo hardware erreboluzio bat, prozesagailu kuantikoak adibidez) honek iraultza bat suposa dezake eguneroko bizitzan. Milaka gauza automatizatzeko aukera egongo litzateke, adibidez kotxeak gidatzen duten algoritmoak. Honek istripu kopurua jaitsi dezake, edo itsu batek lehen aldiz ez luke laguntzarik beharko kotxean mugitzeko. Desgaitasun bat duten pertsonek robot bidezko asistentzia izateko aukera edukiko lukete arazo konkretu batzuekin ere.

Algoritmo zahar eta modernoen diferentzia nagusia moldakortasuna dela ikusi da. Sareak beste joko batzuekin ondo ibiltzea espero da, oinarrizko q-learning algoritmoak berriz arazo gehiago izango luke joko konplexuekin. Tetris jokoa adibidez, ingurunea ez da determinista eta honen ondorioz zuhaitza zabalegia izango litzateke esplorazioa ona izateko eta ikasitako informazioa erabiltzeko. Gainera, lortzen den saria denboran nahiko atzeratuta dago. Sareak jokoaren eredua ikastea espero da, emaitza hobeak lortuz.

Bibliografia

[1] E. Stevens, L. Antiga, T. Viehmann (2020): Deep Learning with PyTorch. https://pytorch.org/deep-learning-with-pytorch.

[2] C. C. Aggarwal (2018): Neural Networks and Deep Learning.

[3] A. Géron (2019): Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow.

[4] I. Vasilev Et Al. (2019): Python Deep Learning, second edition.

[5] R. S. Sutton, A. G. Barto (2014): Reinforcement Learning: An Introduction.

[6] A. Nandy, M. Biswas (2018): Reinforcement Learning With Open AI, TensorFlow and Keras Using Python.

[7] V. Mnih Et Al. (2013): Playing Atari with Deep Reinforcement Learning. https://arxiv.org/pdf/1312.5602.pdf.

[8] S. Ravichandiran (2018): Hands-On Reinforcement Learning with Python.

[9] M. Lapan (2018): Deep Reinforcement Learning Hands-On

[10] M. Hessel Et Al. (2017): Rainbow: Combining Improvements in Deep Reinforcement Learning. https://arxiv.org/pdf/1710.02298.pdf

[11] C. Berner Et Al. (2019): Dota 2 with Large Scale Deep Reinforcement Learning. https://cdn.openai.com/dota-2.pdf

Eranskinak

Mario jokoa exekutatzerako garaian RAM atzipena oso baliagarria izan da jokoa kontrolatzeko. Adibidez Mario noiz hil den jakiteko, edo garraiaturiko distantzia atzitzeko. Jokoaren bi memoria erabili dira: RAM memoria, eta CIRAM (informazio grafikoa duena). RAM memoriaren edukia detektatzeko lagungarria izan da web orrialde hau[11].

CIRAM memoria

Algoritmoa Python erabili gabe probatzeko (Lua bakarrik, dependentziarik gabe) RAM memoriatik informazio grafikoa atzituko da. Kontsolaren CIRAM (nametable) memorian aurkitzen da informazio hau, baina pixel balioak beharrean tile balioak (8x8 pixel) gordetzen dira. Hasiera batean hau erabili da. CIRAM memoria horrelako itxura dauka:

Figure 31. CIRAM memoria

[11] https://datacrystal.romhacking.net/wiki/Super_Mario_Bros.:RAM_map

Lau pantaila gordetzeko informazioa du, baina bi bakarrik erabiltzen dira. Kontsolaren memoria mugak direla eta, jokoa horizontala denean beherako bi pantailak gorakoaren kopiak dira. Jokoa bertikala denean eskuineko bi pantailak ezkerrekoen kopia bat da, baina Marion hau ez da inoiz gertatzen, maila guztiak horizontalak direlako.

Diseinu honekin, Mario dagoen pantaila kargatzen da, eta ondoren erakutsiko den informazioa kargatzen da. Mario eskuinera mugitzen doan eñean, CIRAM memoriak hurrengo tile balioak kargatzen ditu zutabeen gainean idazten.

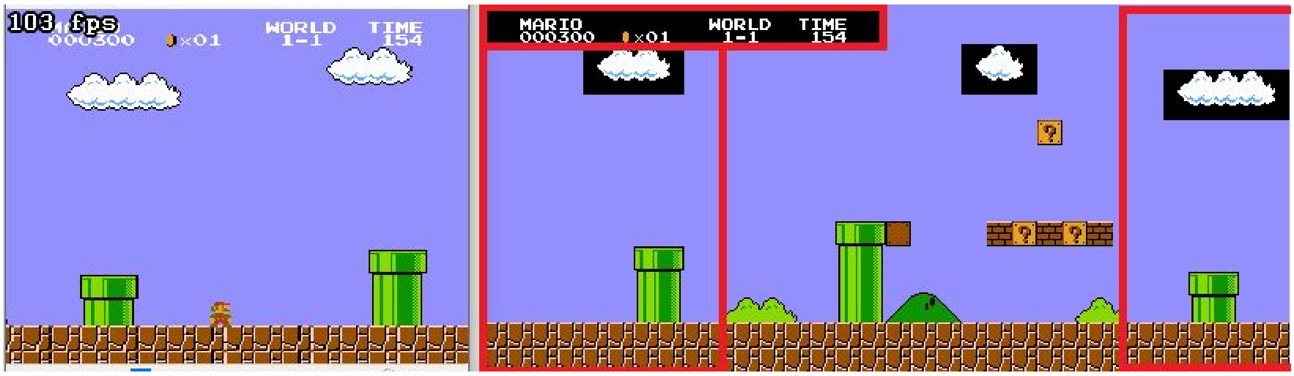


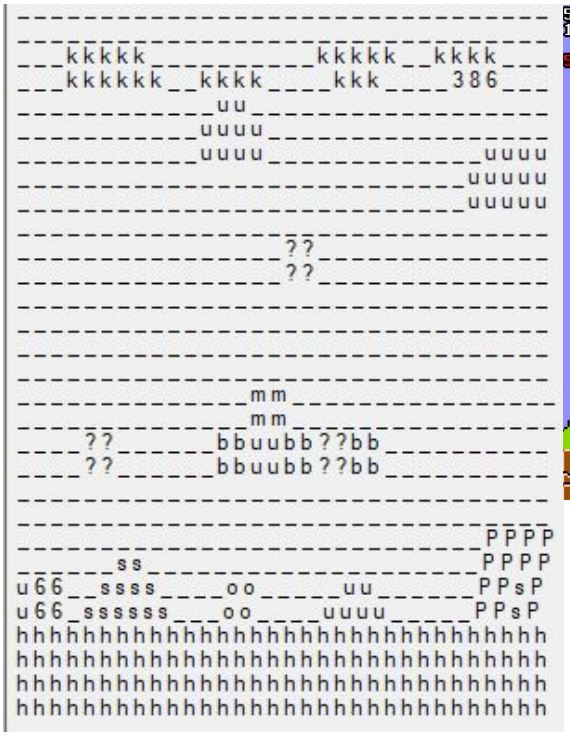
Figure 32. Viewport aldakorra CIRAM memorian zehar

CIRAM taulan bi pantaila erabiltzen dira, 32x30 tamaina badu pantaila batek, honek memorian 64x30 byte segida osatzen du. Mario mugitzen doan eñean, kargatuko den espazio berria taulan agertzen da, atzean zegoen espazioarekin ordezkatuz (gaineko 4 lerroak izan ezik, puntuazioa, txanponak eta denbora balioak ez dira mugitzen). Taulak honetan "tile" balioak gordetzen dira. Tile bat 8x8 pixelez osatutako pantailaren zati bat da. Balio hauek pantailan erakutsi ondoren, itxura hau du:

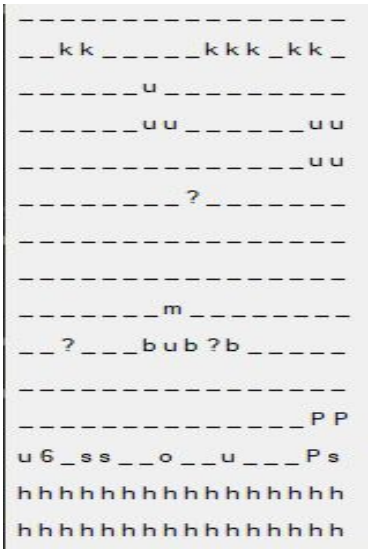
```
36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36
36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36
36 36 36 22 10 27 18 24 36 36 36 36 36 36 36 36 36 36 32 24 27 21 13 36 36 29 18 22 14 36 36 36
36 36 36 0 0 0 0 0 0 36 36 46 41 0 0 36 36 36 36 1 40 1 36 36 36 36 3 8 2 36 36 36
36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36
36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36
36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 54 55 36 36 36 36 36 36 36 36 36 36 36 36
36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 53 37 37 56 36 36 36 36 36 36 36 36 36 36 36
36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 57 58 59 60 36 36 36 36 36 36 36 36 36 36 36
36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36
36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36
36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36
36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36
36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36
36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36
36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36
36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36
36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36
36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36
36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36
36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36
36 36 36 36 49 50 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36
36 36 36 48 38 52 51 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36
36 36 48 38 38 38 38 51 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36
36 48 38 52 38 38 52 38 51 36 36 36 36 36 36 36 36 36 36 36 36 36 36 36 54 55 54 55 54 55 36 36
48 38 38 38 38 38 38 38 38 51 36 36 36 36 36 36 36 36 36 36 36 36 36 53 37 37 37 37 37 37 56 36
180 181 180 181 180 181 180 181 180 181 180 181 180 181 180 181 180 181 180 181 180 181 180 181 180 181 180 181 180 181 180 181
182 183 182 183 182 183 182 183 182 183 182 183 182 183 182 183 182 183 182 183 182 183 182 183 182 183 182 183 182 183 182 183
180 181 180 181 180 181 180 181 180 181 180 181 180 181 180 181 180 181 180 181 180 181 180 181 180 181 180 181 180 181 180 181
182 183 182 183 182 183 182 183 182 183 182 183 182 183 182 183 182 183 182 183 182 183 182 183 182 183 182 183 182 183 182 183
```

36 zenbakia zeruaren identifikatzailea da adibidez, eta 180, 181, 182 eta 183 lurreko blokea osatzen dituen lau aldeak.

CIRAM memoria esker atzealdearen informazioa lor daiteke, baina arazo bat dago: sprite-ak ez dira agertzen. Sprite bat entitate bat da jokoaren barruan, posizio dinamikoa duena. Adibidez Mario, etsaiak, power-up edo suzko bolak. Balio hauek lortzeko RAM helbideak begiratu dira, informazioa atzitzeko eta sortutako arrayean dagokion lekuan agertzeko. Honetarako X posizioa, Y posizioa, existitzen den ala ez, Marioren egoera (txikia/handia), eta horrelako hainbat balio atzitu dira, kalkuluak egin jokoaren “viewport” barruan dauden ala ez jakiteko, eta barruan badaude pantailako posizioa kalkulatu behar da.
X posizioa kalkulatzeko bi balio behar dira, p1 eta p2, RAM memorian daudenak. Balio hauek [0, 255] tartean egongo dira, eta p1 256-ra iristen denean p2 = p2 +1 izango da, eta p1-ek 0 balioa izango du berriro. Beraz, posizio totala p2 * 256 + p1 izango da.
Hau saria kalkulatzeko ere erabiliko da, eta ez bakarrik Mario gure egoeran agertzeko.



Entitateak pantailaratzean: Mario ‘o’ letraz osatuta dago, txanpiñoia ‘m’ eta etsaia ‘6’. Txukunago ikusteko zenbakiak letrengatik aldatu dira.



Bloke bat 2x2 tilez osatuta dago. Lurra osatzen duen bloke batek adibidez, 180 181 182 eta 183 tileak erabiltzen ditu. Honek aukera ematen du 32x30tik 16x15-ra jaisteko eta blokeen informazioa gordetzeko tileen ordez.

32x30 beharrean 16x15 erabiliz zehaztasun asko galtzen da, entitateen posizioa oso mugatua dagoelako.

RAM memoria

Memoria hau datu batzuk atzitzeko erabili da. Ez dira agertuko tile informazioa osatzeko bakarrik erabili diren datuak (adibidez etsaien posizioa edo zein motakoak diren). Helbide bakoitzak zer informazio gordetzen duen jakiteko web horri hau [12]erabili da laguntza moduan.

- Marioren X posizioa: Saria kalkulatzeko.
- Marioren Y posizioa: Zulotik behera erortzen denean partida berria hasteko, hainbat segundo itxaron gabe.
- Marioren egoera gordetzen duen helbidea: Mario hiltzen denean detektatu eta partida azkar berrabiarazteko erabiliko da.
- Mariok bandera ukitzen dagoen gordetzen duen helbidea: Maila pasatzen denean detektatzeko erabiltzen da.
- Zein mailan dagoen: Bi helbide dira, eta mailaren informazioa gordetzen da (4-2 adibidez). Informazio hau ur mailak detektatzeko erabiltzen da (2-2 eta 7-2 mailak), eta kasu horretan ekintzak ez dira ekiprobableak izango q-learning algoritmoan.

Geruza motak eta operazioak

Guztiz konektaturiko geruza

Geruza mota hauetan neurona bakoitza hurrengo geruzako neurona guztiekin dago konektaturik. Azken kapak topologia hau izango du, zein izango den egin beharreko ekintza aukeratzen du azken geruzak (balio altuena hoberena da). Keras liburutegian dense izena dute, eta pytorchen linear.

Geruza hauek X input kopuruko neuronak Y kopurura jaisteko erabili daitezke, konexio eta output gutxiago izateko. Hau lortzeko transformazio lineal bat aplikatzen da. Demagun 100x1 tentsore bat dugula geruza baten sarreran, 4x100 matrize pisudun tentsore bat erabili daiteke matrize biderketa egiteko, 4x100 * 100x1 = 4x1 tamainako irteera lortuz. y = Ax + b formula aplikatzen da, non:
y = irteerako tentsorea
x = sarrerako tentsorea
A = pisudun matrize tentsorea
b = bias tentsorea (posiblea da hau ez erabiltzea)

Konboluzio geruzak

Geruza honek pixel multzoak eskaneatzen ditu, irudiaren ezaugarriak antzemateko asmoz. Adibidez, konboluzio kapa bat erabili daiteke irudi batean objektuak antzemateko (pertsonak, hegazkinak…).

[12] https://datacrystal.romhacking.net/wiki/Super_Mario_Bros.:RAM_map

Geruza hauetan gertu dauden pixelak hurrengo geruzako neurona berdinera daude konektaturik, hau da, ez da guztiz konektaturiko geruza bat.

Geruza hauek bi parametro erabiltzen dituzte: Kernel, edo filtroaren tamaina, eta stride, zenbat mugituko den (honek datu kopurua jaisten du ere, eta ez denean erabiltzen defektuzko balioa 1 da).

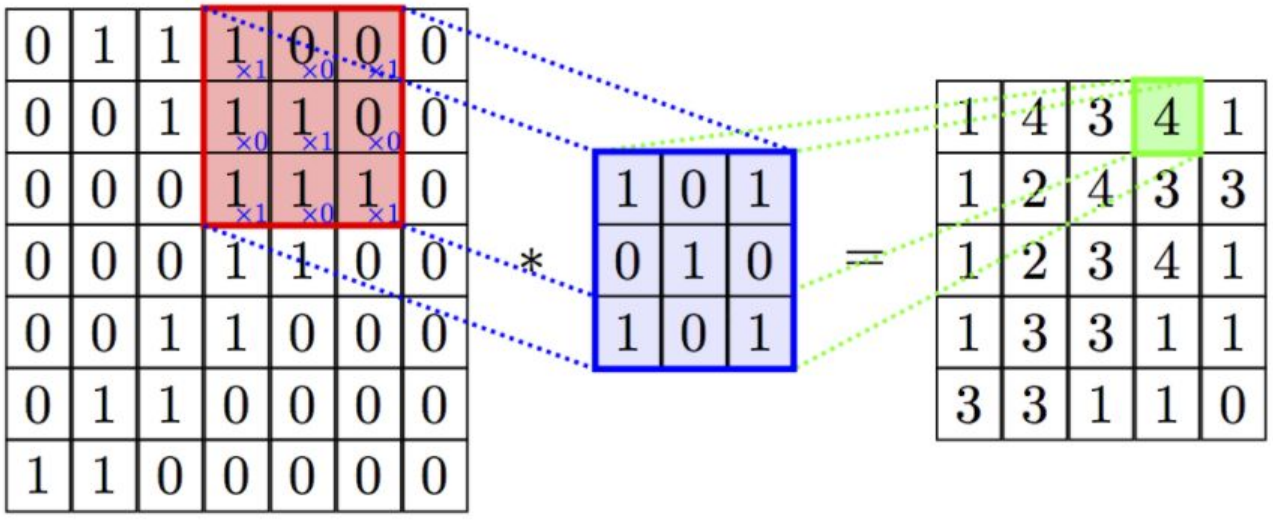


Figure 33. Konboluzio eragiketa

Filtro hauekin eredu espazialak detektatuko ditu sareak. Irudiak manipulatzeko garaian oso filtro interesgarriak aurkitu daitezke, batzuek argazkia difuminatzeko balio dute, bestek batzuek ertzak detektatzeko, edo irudia zorrozteko. Gure kasuan kernelaren baloreak entrenatuko diren pisuak izango dira.

Hasieran kernel tamaina handiak erabiltzen dira (8x8) obektu handiak detektatzeko, eta gero txikiagoak (4x4 eta 3x3), txikiagoak diren detaileak antzemateko.

Aktibazio funtzioa

Neurona guztiek, azkeneko geruzak izan ezik, aktibazio funtzio bat erabiltzen dute.

Lineala ez den aktibazio funtzio bat erabiliko da, hauek hainbat abantaila dituzte:
Hainbat neurona geruza pilakatzea onartzen du deep sare neuronal bat sortzeko.
Eredu konplexuak antzeman eta sortzeko beharrezkoa da lineala ez den funtzio bat erabiltzea.
Arrisku gutxiago dago vanishing gradient arazoarekin topatzeko,

Konboluzio geruzetan ohikoena da ReLU (Rectified Linear Unit) funtzio bat erabiltzea. Funtzio hau konputazionalki merkea da, eta balio negatiboak zero bihurtzen ditu.
f(x) = max (0, x) da funtzioaren definizioa, eta honen balore posibleak [0, ∞) tartean daude.
Posiblea da bias parametroa erabiltzea atari moduan, 0 beharrean beste edozein zenbaki aukeratzeko.

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

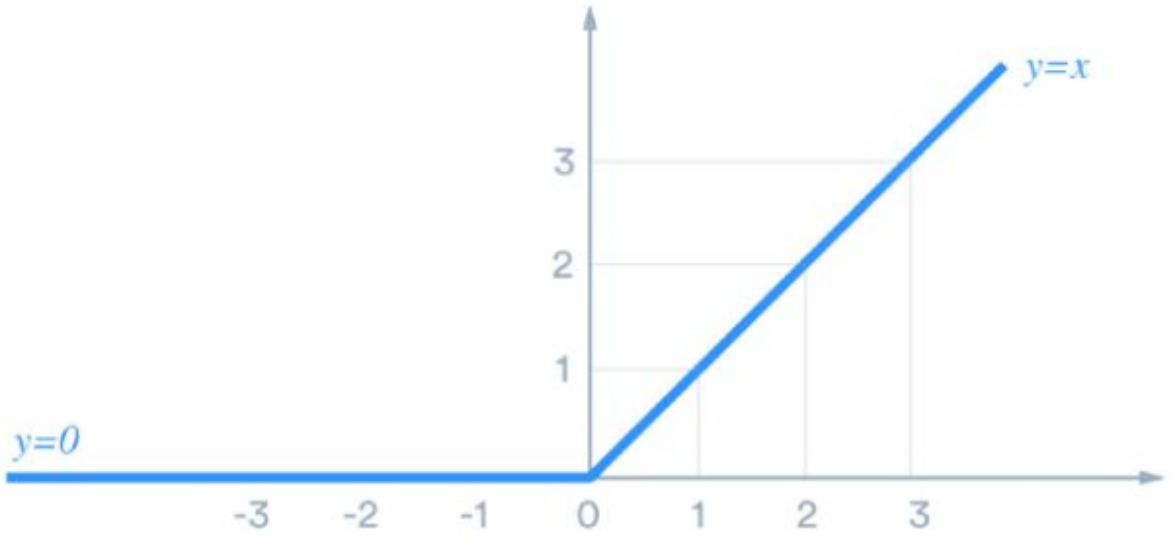
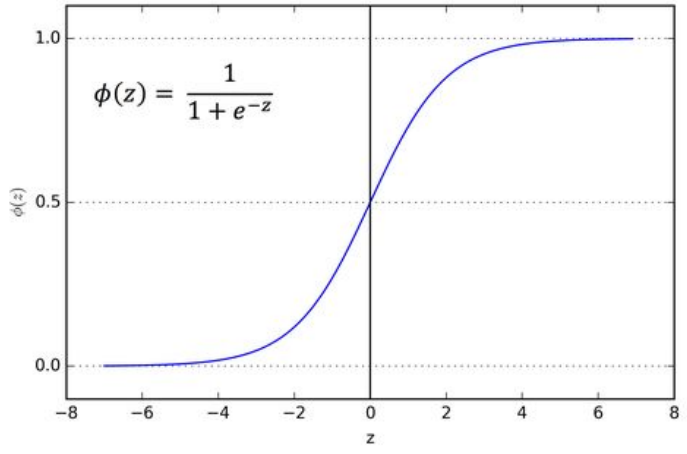


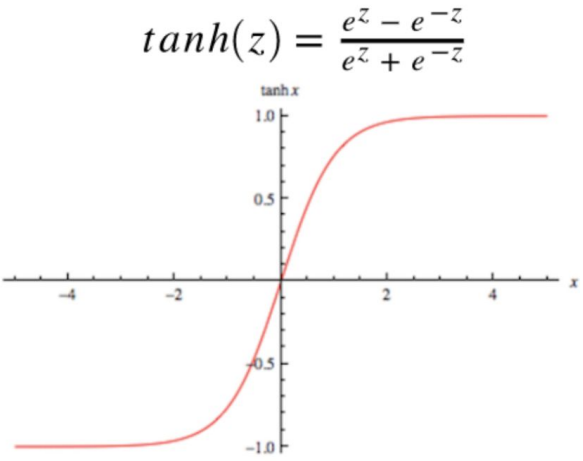
Figure 34. ReLU aktibazio funtzioa

ReLU-z gain beste aktibazio funtzio batzuk ere daude, ezagunenak hauek dira:

Sigmoid funtzioa, funtzio honen balioak (0, 1) tartean daude. Probabilitateak kalkulatzeko aukera ona da, baina ikasketan arazoak sortu dezake “vanishing gradient” arazoarengatik.

Tanh funtzioa sigmoid funtzioaren antzekoa da, S forma du baina baloreak (-1, 1) tartean kokatzen dira. Funtzio hau bi elementuko sailkatzaileetan erabiltzeko aukera ona da (katua ala txakurra agertzen den argazki batean, adibidez).





Softmax funtzioa ere batzuetan erabiltzen da, normalean sarearen azkeneko geruzan. Honek irteerako balio guztiak [0, 1] tartera mugatzen ditu, eta balore guzti hauen batuketaren emaitza 1 izango da. Hau aukera ona da hainbat aukerako sailkatzaile bat sortzeko eta probabilitateak kalkulatzeko, baina Marioren arazoa erregresio arazo bat denez azken geruzan ez da aktibazio funtziorik erabiltzen.

Erabiltzaile gida

Atal honetan proiektua exekutatzeko eman behar diren pausoak azaltzen dira.

Python erabiltzeko dependentziak instalatu behar dira, pip install komandoa oso erosoa da hau egiteko.

Q-learning bakarrik erabili nahi bada PIL image liburutegiarekin nahikoa da. Sarea erabiltzeko hainbat dependentzia instalatu behar dira:

Sare neuronala erabiltzeko, hainbat liburutegi gehiago behar dira:
-Numpy
-Pytorch. CUDA ere instalatu behar da, GPU-a erabiltzeko.
-Gym
-Matplotlib

httpserver.py fitxategiko lerro bat aldatu behar da, erabiltzaileak erabiliko duen IP helbidea eta portua zehaztuz.

```
httpd = HTTPServer(('192.168.0.13', 8081), SimpleHTTPRequestHandler)
```

Aldaketa hau egin ondoren, Python zerbitzaria exekutatzeko prest dago.

pipe.lua fitxategiaren barruan dagoen getHttp() funtzioak erabiltzen duen IP helbide eta portua aldatu behar da ere:

```
image = comm.httpPost("http://192.168.0.13:8081","clipboard")
```

EmuHawk exekutatzeko terminal bidez exekutatu behar da, argumentu moduan IP helbide eta portua pasaz:

```
.\EmuHawk.exe --url_post=http://192.168.0.13:8081
```

Hau egin ondoren jokoa eta Lua script-a exekutatu behar dira:

File -> Open ROM erabiliz Mario jokoa ireki behar da.
Tools -> Lua Console gaitu, eta ondoren Open Script ikonoa erabili behar da programa.lua kargatzeko. Toggle Script ikonoa erabiliz programa exekutatzen hasiko da.

Defektuz hirugarren savestate-a erabiliko da, beraz State karpetako fitxategi bat kargatu behar da (File->Load State->Load Named State) eta ondoren hirugarren savestate posizioan gorde (File->Save State->3). Hau egiteko emulagailua pausatu behar da. Honek definitzen du Mario non hasiko den hil eta gero (maila baten hasiera izango da).

Sare neuronala erabiltzeko httpserver.py erabili beharrean httpserverNN.py erabili behar da, aurreko kasuan bezala IP helbidea aldatzen, eta programaNN.lua erabili behar da programa.lua beharrean.

Sarearen konputazio grafoa

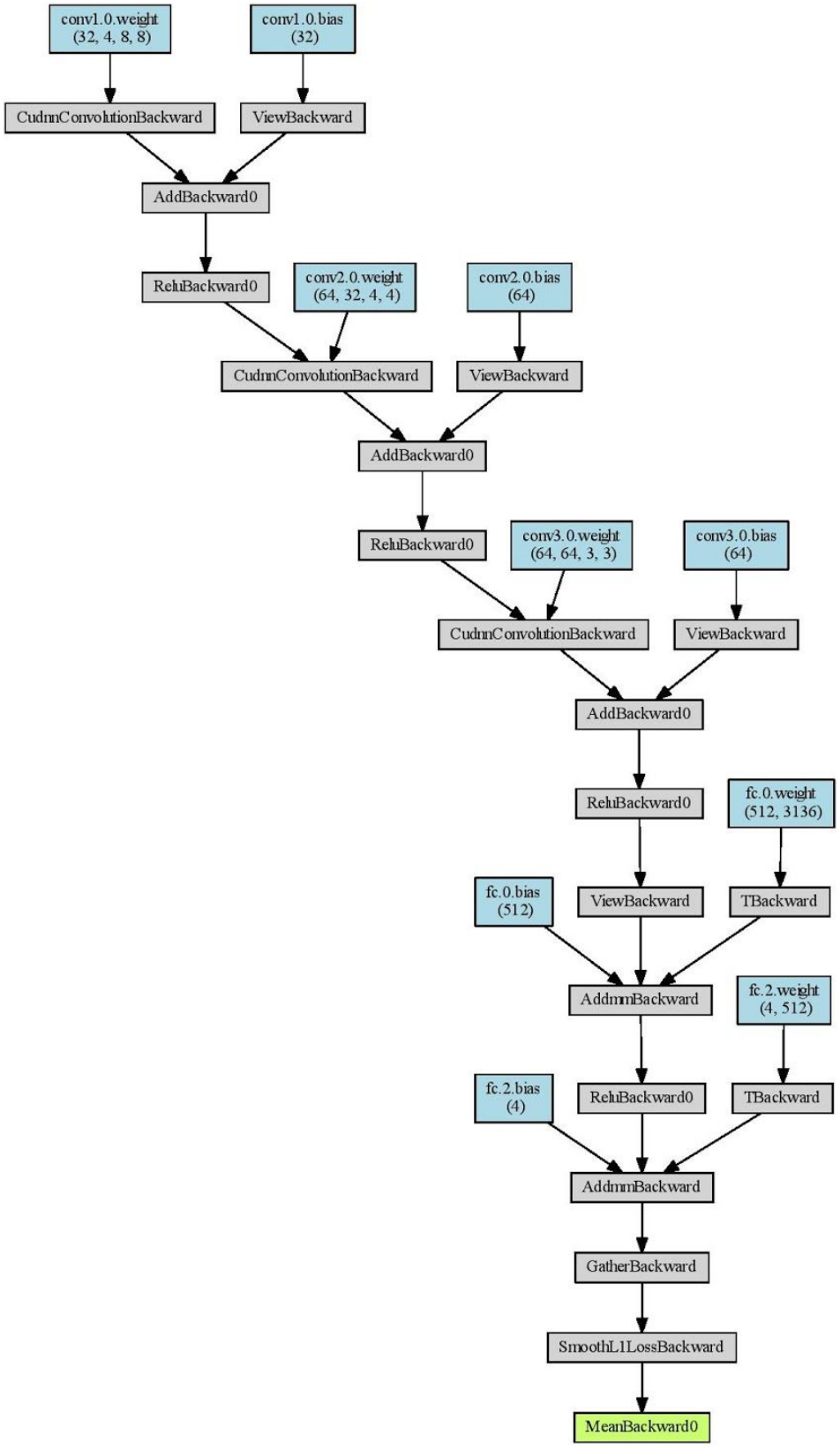


Figure 35. Konputazio grafoa